HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

Diplomarbeit

Herr
Stephan Reuther

TYPO3-Extension für einen Bestelldienst - Entwurf und Realisierung

2010

Fakultät **Informationstechnik & Elektrotechnik**

Diplomarbeit

TYPO3-Extension für einen Bestelldienst - Entwurf und Realisierung

Autor:
Stephan Reuther

Studiengang:
Multimediatechnik

Seminargruppe:
MK05w2

Erstprüfer:
Prof. Dr.-Ing. habil. em. Lutz Winkler

Zweitprüfer:
M.S.c. Dipl.-Ing. (FH) Rico Thomanek

Chemnitz, Juni 2010

Bibliographische Beschreibung

Stephan Reuther:
TYPO3-Extension für einen Bestelldienst - Entwurf und Realisierung. - 2010. - 85 S.
Chemnitz, Hochschule Mittweida, Fakultät Informationstechnik & Elektrotechnik, Diplomarbeit, 2010

Referat

Das Ziel dieser Diplomarbeit ist der Entwurf und die Realisierung eines Bestelldienstes als Webanwendung auf der Basis von TYPO3 4.3+. Die Arbeit ist eine Synthese aus bewährten Methoden und empirischer Arbeit an bisher nahezu undokumentierten Softwarebestandteilen. Es wird gezeigt, wie moderne Extensionprogrammierung für TYPO3 aussehen kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface (Programmierschnittstelle)
CMS	Content Management System
CSS	Cascading Stylesheets
DDD	Domain-driven Design
GUI	Graphical User Interface (Grafische Nutzeroberfläche)
HTML	Hypertext Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
MVC	Model-View-Controller
OOP	Objektorientierte Programmierung
PHP	PHP Hypertext Preprocessor
SQL	Structured Query Language
SVN	Subversion
TCA	Table Configuration Array (TYPO3)
(U)ID	(eindeutiger) Identifikator, engl. (unique) Identifier
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation

Essen ist eine elementare Tätigkeit. Die kulinarische Versorgung spielt besonders bei sozialen Einrichtungen eine wesentliche Rolle, denn: diesen Einrichtungen, seien es Krankenhäuser oder Kindergärten, sind Menschen anvertraut, für deren Wohlbefinden sie Verantwortung tragen. Die Verantwortung soll dabei positive Gefühle bei den Betreuern hervorrufen. Nur wie schnell kann einem ein ungeordneter oder komplizierter Arbeitsablauf die Verantwortung über den Kopf wachsen lassen? Wenn man bei aller Hektik – es gibt ja solche Tage – dann schnell noch das Fax mit der Speisebestellung fertig machen muss, weil sonst die zuliefernde Küche nicht mehr rechtzeitig liefern kann. Und dann sind auch noch die Bestellzettel alle.

In solch sensiblen Bereichen müssen die Arbeitsabläufe optimiert und den Betreuern so einfach wie möglich gemacht werden. Mit der Computertechnik haben wir eine großartige Möglichkeit Informationen schnell und unkompliziert über Netzwerke zu verschicken. Mit Website-Formularen kann man übersichtlich jede gewünschte Information erfragen und das beste ist – Das Papier geht nie aus. Mit einem internetfähigen Computer sind alle Voraussetzungen erfüllt, um einen Bestellservice lauffähig zu bekommen. Über eine ansprechende Nutzeroberfläche können alle Speisen bestellt werden, die die Küche zu bieten hat.

Die sozialen Einrichtungen sollen mit dem Bestellservice ein Werkzeug in die Hand bekommen, das ihnen den Dienst am Menschen erleichtert. Dieser Dienst soll Freude bereiten und nicht in Stress ausarten.

1.2 Hintergrund

Diese Arbeit verfolgt mehrere Ziele. Im Zentrum steht natürlich die Umsetzung des Bestelldienstes nach den Vorgaben des Auftraggebers. Da ich mich persönlich schon seit einigen Jahren mit der Extensionentwicklung für TYPO3 auseinandersetze, möchte ich in dieser Arbeit meinen aktuellen Erkenntnisstand in diesem Bereich dokumentieren. Aufgrund des hohen Anteils persönlicher Erfahrungen in dieser Arbeit, habe ich bewusst die Ich-Erzählperspektive für Beschreibungen gewählt.

Seit meinem Besuch der T3DD09 (TYPO3 Developer Days 2009) habe ich mich intensiv mit Extbase und Fluid auseinandergesetzt. Diese beiden Module, in TYPO3 Version 4.3 neu dazugekommen, sind selbst zum heutigen Zeitpunkt kaum dokumentiert[1]. Meine Recherchearbeit und die Suche nach dem optimalen Einsatz dieser beiden Module soll in dieser Arbeit einen wichtigen Platz haben. Da es kaum gedruckte Literatur zu den sich auch zum jetzigen Zeitpunkt noch weiterentwickelnden Modulen gibt, kann ich dafür als Quelle hauptsächlich Videovor-, Blogein- und Mailinglistenbeiträge heranziehen. Wo es geht habe ich mich jedoch bemüht auch gedruckte Literatur zu Rate zu ziehen.

Ich werde bei der Beschreibung der Extensionentwicklung viel Wert auf die neuen Technologien legen und setze grundlegende Kenntnis des TYPO3-Systems voraus, bzw. betrachte es als nicht elementar für diese Arbeit.

Ich möchte an dieser Stelle all denen danken, die zur Entstehung dieser Arbeit beigetragen haben. Meiner Frau, meinen Eltern und Geschwistern und natürlich auch den Betreuern an der Hochschule Mittweida, Prof. Dr.-Ing. habil. em. Lutz Winkler und M.S.c. Dipl.-Ing. (FH) Rico Thomanek.

1 Pünktlich nach Abgabe dieser Diplomarbeit, nämlich am 5. Juli 2010 erscheint das Buch: „Zukunftssichere TYPO3-Extensions mit Extbase und Fluid“ von Jochen Rau und Sebastian Kurfürst bei O'REILLY. Die Autoren sind die Chefentwickler der beiden Module.

1.3 Kapitelübersicht

Die Arbeit so aufgebaut, wie auch ein Projekt chronologisch ablaufen würde.

So werden im 2. Kapitel zunächst die Anforderungen an den Bestellservice beschrieben. Danach werden gängige Praktiken vorgestellt und demonstriert, mit deren Hilfe man eine hochwertige Dokumentation und Einstiegshilfe für andere am Projekt Beteiligte erstellen kann. Die Beschreibung im 2. Kapitel ist unabhängig von der Plattform, auf welcher der Bestellservice später umgesetzt wird.

Es folgt eine Beschreibung der einzusetzenden Technologien und Werkzeuge im 3. Kapitel. Es handelt sich dabei um die Technologien, die durch die Anforderungen an die Umsetzung des Projektes vorgegeben sind. Besonders ausführlich werden das Extbase-Framework und die Template Engine Fluid erläutert.

Daran anknüpfend werden im 4. Kapitel wesentliche Konzepte bei der Umsetzung des Bestellservices vorgestellt. Anhand von Beispielen werden Probleme und Lösungen mit den eingesetzten Technologien beschrieben.

Die Arbeit schließt im 5. Kapitel mit einem **Fazit**, welches die Erkenntnisse der Arbeit zusammenfasst und einen Ausblick bietet.

2 Konzeption des Bestellservice

Das Vokabular dieser Arbeit stammt aus den Bereichen IT und Webprogrammierung. Die Kenntnis von Begriffen und Vorgängen im IT-Bereich, der objektorientierten Programmierung und Ausdrücken aus der TYPO3-Welt wird teilweise vorausgesetzt. Zur weiteren Vertiefung können die Quellen im Literaturverzeichnis genutzt werden.

Häufig verwendete Fremdworte werden im folgenden mit einer Erklärung eingeführt. Die Beschreibungen sind meist projektnah und können von denen anderer Quellen abweichen.

2.1 Projektbeschreibung

2.1.1 Anforderungen an den Bestelldienst

Der Bestelldienst wird von einem Serviceunternehmen bereitgestellt, das an unterschiedlichen Orten Küchen betreibt. Diese bieten ortsansässigen Unternehmen Speisenversorgung und Catering an.

Die Nutzer des Bestelldienstes sind überwiegend soziale Einrichtungen, wie Krankenhäuser oder Kindergärten, denen das Wohl, die Gesundheit und die Erziehung anderer Menschen anvertraut sind. Und gerade dort ist die kulinarische Versorgung von elementarer Bedeutung. Deswegen sollte die Bestellung einfach, schnell und reibungslos funktionieren. Da sich die wenigsten Unternehmen extra eine Person leisten können, die ausgiebig die Speisebestellung durchführt und dafür eine Menge Zeit hat, sollte die Bestellung auch von einer gestressten Krankenschwester erfolgreich durchgeführt werden können, bevor sie durch das dringende Klingeln eines Notfallpatienten wieder vom Rechner weggeholt wird.

Es gibt 3 verschiedene Nutzergruppen, die den Bestellservice nutzen können und die ich hier beschreiben möchte:

Lieferstelle

Sogenannte Lieferstellen melden sich am System an und geben in ein vorgefertigtes Formular ihre Bestellung ein. Es gibt zwei kombinierbare Arten, was bestellt werden kann. Zum einen können im Bestellformular Personen angegeben werden, die an bestimmten Mahlzeiten teilnehmen, zum anderen können auch einzelne Kostformen bestellt werden. Die Kostformen können für eine übersichtlichere Darstellung in Kategorien eingeteilt werden. Jede Kostform hat eine bestimmte Einheit (z.B. Stück oder Personen), die beim Erstellen der Kostform angegeben wird. Außerdem können Preise für die Kostformen hinterlegt werden.
Für jede Bestellung gibt der Besteller seinen Namen ein und hat die Möglichkeit, eine kurze Notiz zur Bestellung zu hinterlassen. Beim Abschicken der Bestellung wird diese an einer zentralen Stelle gespeichert. Für jeden Kunden gibt es eine späteste Bestellzeit, die aussagt, bis zu welcher Uhrzeit für welchen Tag bestellt werden kann. Zum Beispiel kann ein Kunde, der für Dienstag bestellen will, dies nur bis Montag 14:00 Uhr tun. Bestellungsänderungen durch die Lieferstellen vor der spätesten Bestellzeit sind möglich. Dabei werden die Werte, die sich im Vergleich zur vorherigen Bestellung geändert haben, deutlich hervorgehoben.
Ein Ausgangsbuch ermöglicht die Einsicht in die aktuelle Bestellung und in die Bestellungen der letzten Tage.

Küche

Wenn sich eine Küche am System anmeldet, kann sie die aktuellen Bestellungen der Lieferstellen einsehen. Sie kann auch Änderungen an Bestellungen vornehmen, deren späteste Bestellzeit schon abgelaufen ist. Somit können kurzfristige Änderungen zwischen Lieferstelle und Küche telefonisch abgesprochen und von der Küche im System angepasst werden. Die Küche erhält die Möglichkeit, die Bestellungsübersicht auszudrucken.

Rechnungsstelle

Die Rechnungsstelle erhält eine monatige Ansicht der Bestellungen ihrer Kunden inklusive dem Einzel- und Gesamtpreis der angegebenen Personen/Kostformen. Die Darstellung erfolgt pro Kunde und Monat und kann ausgedruckt werden.

Das also ist die Projektbeschreibung. Nicht immer hat man das Glück, auf so engem Raum die wichtigsten Fakten zum Projekt beschrieben zu haben. Allerdings lässt die Kürze hier auch noch einige Fragen offen. Andererseits muss ein 50-seitiges Lastenheft auch erstmal durchgearbeitet und analysiert werden, und ob danach die Einzelheiten des Projektes klar sind ...

Auf spezielle Anforderungen an den Bestellservice werde ich zu gegebener Zeit in den jeweiligen Kapiteln eingehen.

2.1.2 Anforderungen an die Umsetzung des Bestelldienstes

Der Bestelldienst soll als TYPO3-Extension[2] umgesetzt werden. Dadurch werden einige Parameter für die Planung von Software schon von vornherein festgelegt. Der wichtigste Punkt für den Anwender ist, dass er mit sehr wenigen Komponenten Zugriff auf den Bestelldienst hat. Er braucht dafür nur einen internetfähigen PC mit einem installierten Browser (z.B. Firefox, Internet Explorer) und da die meisten Büros mit Internet und Rechner ausgestattet sind, kommen auf die Nutzer des Bestelldienstes keine zusätzlichen Kosten für Wartung und Installation zu.

Der Auftrag zur Umsetzung des Bestelldienstes läuft über development.IT, einer Agentur, für die ich nun schon seit einiger Zeit arbeite und über die ich zu TYPO3 gekommen bin. Bei der Arbeit für development.IT habe ich auch direkte Erfahrung mit den Kunden sammeln und einige Projekte von der ersten Skizze bis zur Übergabe begleiten können.

2 Siehe *3.3.1 Das TYPO3 Content Management System*

2.2 Usecases

Die Projektbeschreibung des Bestelldienstes aus 2.1.1 Anforderungen an den Bestelldienst ist eine umfangreiche Beschreibung des Gesamtproblems, das es zu lösen gilt. Und große Probleme sind meistens nur zu lösen, wenn man sie in hinreichend kleine Teilprobleme zerlegt. Diese Teilprobleme lassen sich sehr gut in Usecases (Anwendungsfälle) ausdrücken. Usecases beschreiben Teile des Systems und ihre Verwendung. Nicht unbedingt aus der Sicht der Nutzeroberfläche. Sie können auch noch abstrakter definiert werden.

In allen Phasen des Projektes können Usecases als Diskussionsgrundlage für die am Projekt Beteiligten genutzt werden. Dadurch, dass sie das Projekt in allen Einzelheiten beschreiben, kann anhand der Usecases auch geprüft werden, ob das fertige Projekt den Anforderungen genügt. Fehlverhalten des Systems kann mit Bezug auf einen Usecase unmissverständlich kommuniziert und dadurch auch schneller beseitigt werden.

Im Beispiel des Bestellservices resultieren die Usecases aus der Projektbeschreibung. Es könnte aber auch die Projektbeschreibung aus der Summe der Usecases erstellt werden. Je nachdem, ob der Kunde mit einem fertigen Konzept anklopft, oder man das Konzept mit dem Kunden erarbeitet, wird der eine oder andere Weg beschritten. Auf Usecases sollte man in keinem Fall verzichten. Sie tragen zur Qualitätssicherung des Projektes bei.

Es gibt verschiedene Arten Usecases darzustellen. Je komplexer das Projekt und die Anwendungsfälle sind, umso mehr kann und sollte man ins Detail gehen[3]. Für den Bestellservice reicht mir eine einfachere Definition. In den beiden folgenden Unterabschnitten zeige ich die Darstellung der Usecases für den Bestellservice in Prosa- und Tabellenform. [dpp][4] (S. 192 ff.)

3 Ein ausführliches Beispiel zeigt [so07] S. 193 f.

4 Die Zeichenfolgen in eckigen Klammern sind Schlüssel für das Fehler: Referenz nicht gefunden

2.2.1 Usecase-Beschreibung

#01: Login

Da es sich um ein nutzerbasiertes System handelt, also verschiedene Rollen und Rechte existieren, sollen die Nutzer über Loginname und Password eindeutig identifiziert werden. Dafür muss ein Formular bereitgestellt werden, in das die Anmeldedaten eingegeben werden können.

#02: Logout

Natürlich soll es auch die Möglichkeit zum Abmelden aus dem System geben.

#03: Lieferstelle führt eine neue Bestellung durch

Wenn man als Lieferstelle angemeldet ist, soll man eine Bestellung aufgeben können. Dabei wird automatisch geprüft, für welchen Liefertag die Bestellung ausgeführt werden kann. Für diesen Liefertag könnte schon eine Bestellung vorliegen, darum wird zu diesem Zeitpunkt automatisch zwischen *Neubestellung* und *Änderungsbestellung* (siehe **#05**) unterschieden. Die Lieferstelle bekommt ein Bestellformular angezeigt, das ausgefüllt werden kann. Bei Falscheingabe erscheinen aussagekräftige Fehlermeldungen im Formular. Wenn kein Fehler auftritt wird nach dem Senden die aktuell gespeicherte Bestellung angezeigt.

#04: Lieferstelle will *a)* aktuelle Bestellung oder *b)* eine Bestellung der letzten Tage ansehen

Auf einer Übersichtsseite soll eine kompakte Übersicht der aktuellen Bestellung, ein sogenannter *Teaser* (Aufmerksamkeitserreger), angezeigt werden. Wenn eine aktuelle Bestellung vorliegt, kann sie also direkt auf der Startseite eingesehen werden (entspricht *a*). Zudem soll es ein Ausgangsbuch geben, in dem die Bestellungen der vergangenen 7 Tage eingesehen werden können (entspricht *a* und *b*).

#05: Lieferstelle will aktuelle Bestellung bearbeiten

Wenn für das aktuelle Lieferdatum schon eine Bestellung existiert, gilt das Durchführen einer Bestellung als Änderungsbestellung. Das Bestellformular wird angezeigt und die Werte der ursprünglichen Bestellung schon voreingetragen. Lediglich der Bestellername wird nicht übernommen. Das Formular verhält sich bei Falscheingaben wie bei der Neubestellung (**#03**). Bei erfolgreichem Senden wird aktuell gespeicherte Bestellung angezeigt. Im Vergleich zur ursprünglichen Bestellung veränderte Werte sind kenntlich gemacht.

#06: Küche will aktuelle Bestellungen der Lieferstellen eines Kunden sehen

Küche meldet sich an und wählt, wenn ihr mehr als ein Kunde zugeordnet ist, den gewünschten Kunden aus einer Liste aus. Daraufhin werden der Küche die Bestellungen der Lieferstellen des Kunden für den aktuellen Liefertag übersichtlich präsentiert. Über eine Auswahl des Lieferdatums können auch die Bestellungen anderer Tage eingesehen werden.

#07: Küche will aktuelle Bestellungen der Lieferstellen eines Kunden ausdrucken

Die Küche verfährt nach **#06** und wählt Kunde und Lieferdatum aus, deren Bestellungen ausgedruckt werden sollen. Über einen 'Drucken'-Knopf kann der Druckvorgang ausgelöst werden.

#08: Küche will beliebige Bestellung einer Lieferstelle eines Kunden bearbeiten:

Die Küche verfährt nach **#06** und wählt Kunde und Lieferdatum aus, unter denen die zu ändernde Bestellung der Lieferstelle zu suchen ist. Die Bestellungen der Lieferstellen sind aufgelistet und können über einen 'Bearbeiten'-Knopf editiert werden. Dabei erscheint das Formular wie in **#05**, aber der Bestellername ist mit 'Küche' voreingestellt und nicht änderbar. Nach erfolgreichem Absenden des Formulars gelangt die Küche wieder zu ihrer Übersicht aus **#06**.

#09: Rechnungsstelle will monatliche Bestellübersicht sehen

Rechnungsstelle meldet sich an und wählt, wenn ihr mehr als ein Kunde zugeordnet ist, den gewünschten Kunden aus einer Liste aus. Daraufhin wird der Rechnungsstelle die Summe der bestellten Waren der Lieferstellen des Kunden für den aktuellen Monat übersichtlich präsentiert. Zusätzlich werden pro Ware Einzel- und Gesamtpreis ausgegeben. Über eine Auswahl des Liefermonats können auch die Bestellungen anderer Monate eingesehen werden.

#10: Rechnungsstelle will monatliche Bestellübersicht drucken

Die Rechnungsstelle verfährt nach **#09** und wählt Kunde und Liefermonat aus, deren Bestellungen ausgedruckt werden sollen. Über einen 'Drucken'-Knopf kann der Druckvorgang ausgelöst werden.

2.2.2 Usecase-Tabelle

Eine andere Möglichkeit ist, die Usecases in einer Tabelle darzustellen. In jeder Spalte kann der Usecase auf unterschiedliche Art und Weise betrachtet werden, z.B. Vor- und Nachbedingungen, beteiligte Akteure, Normalabläufe und Sonderfälle. Im Vergleich zur Usecase-Beschreibung in Textform ist die tabellarische Darstellung übersichtlicher. Allerdings ist es von den Vorlieben des Teams abhängig, in welcher Art die Usecases präsentiert werden, denn in erster Linie sollen sie, wie alle hier vorgestellten Methoden, nützlich sein, nicht nur standardkonform.

Nachfolgend wird ein ein Beispiel für einen Usecase in tabellarischer Form gezeigt. Die Spaltenbezeichnung habe ich von [so07]. Allerdings könnten diese bei Bedarf auch noch um projektbezogene Kriterien erweitert werden.

Name	Ziel	Vorbedingung	Nachbedingung	Nachbedingung im Sonderfall	Akteure	Normalablauf	Sonderfälle
#003: Lieferstelle führt eine neue Bestellung durch	Eine neue Bestellung ist im System, auf welche Küche und Rechnungsstelle zugreifen können	(1) eingeloggter Nutzer ist eine Lieferstelle (2) es ist gerade Bestellzeit (3) es liegt noch keine Bestellung der Lieferstelle für das Lieferdatum vor	Für das Lieferdatum existiert eine Bestellung, die der Lieferstelle des eingeloggten Nutzers zugeordnet ist	Es existiert keine Bestellung für Lieferdatum UND Lieferstelle	Lieferstelle	- Lieferstelle navigiert zu "neue Bestellung" - Lieferstelle füllt ordnungsgemäß Formular aus, d.h. mindestens Pflichtfelder ausfüllen - Lieferstelle klickt auf "Bestellung verbindlich absenden" - Es erscheint eine Zusammenfassung der Daten der Bestellung - Die Daten werden im System eingetragen, sodass Küche und Rechnungsstelle darauf zugreifen können	(1) Lieferstelle gibt Daten nicht korrekt ein (d.h Pflichtfelder nicht ausgefüllt, falsche Datentypen). Folge: Formularansicht mit Fehlermeldungen und Möglichkeit zur Korrektur der Falscheingaben (2) Lieferstelle bricht Bestellprozess vorzeitig ab, indem im Menü auf andere Seite navigiert wird, bevor Bestellprozess abgeschlossen ist

Abb. 2-1: Usecase-Tabelle (Ausschnitt)

Die komplette Tabelle findet sich im Anhang A Usecase-Tabelle.

2.3 UML-Diagramm der Objekte

Die Unified Modeling Language (UML) ist eine für die Modellierung von Software standardisierte Beschreibungssprache. UML setzt sich zusammen aus modellspezifischen Begriffen und grafischen Elementen, wie Pfeilen und Rechtecken, welche die Beziehung zwischen den Begriffen festlegen. Damit können statische, aber auch dynamische Systeme übersichtlich beschrieben werden. Bei komplexen Systemen ist es manchmal nicht wirklich sinnvoll das gesamte System in einem einzigen Diagramm unterzubringen. Die Übersichtlichkeit leidet unter Umständen darunter. In sich abgeschlossene Teilbereiche können so aber gut dargestellt werden. [php5] (S. 48)

Da es sich bei dem Bestellservice um kein allzu komplexes Projekt handelt, können wir eine Gesamtübersicht wagen:

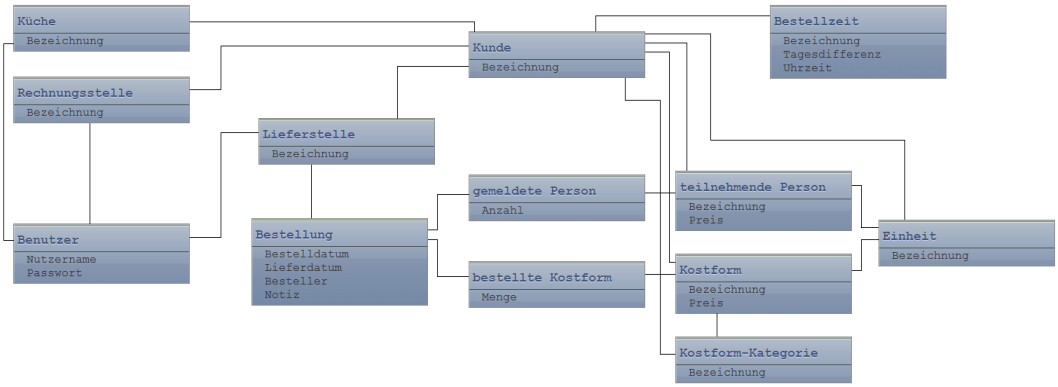


Abb. 2-2: UML-Diagramm der Objekte

Ein **Kunde** (oben in der Mitte) hat eine Bezeichnung. Ihm ist eine beliebige Anzahl an Lieferstellen zugeordnet. Um die Verwaltung für die einzelnen Kunden übersichtlich zu halten besteht die Anforderung, dass für jeden Kunden separat Daten angelegt werden. Dazu gehören die bestellbaren Kostformen und Personenbezeichnungen, die Kostformkategorien und die Einheiten. Ohne diese Anforderungen würden die Abhängigkeiten wegfallen und damit das UML-Diagramm übersichtlicher werden. Damit ginge aber auch die Flexibilität der Anpassung des Systems auf die Verschiedenartigkeit der Kunden verloren, auch hinsichtlich des Preissystems. Einer Küche und auch einer Rechnungsstelle kann jeweils eine beliebige Anzahl an Kunden zur Verwaltung zugewiesen werden.

Bei den **Lieferstellen** dreht sich alles um Bestellungen, von der sie beliebig viele anlegen kann. Außerdem erhält die Lieferstelle eine Bezeichnung.

Eine **Bestellung** als zentrales Element des Bestellservices hat Bestell- und Lieferdatum, den Namen des Bestellers und optional eine Notiz. Zudem sind ihr die gemeldeten Personen und die bestellten Kostformen zugeordnet.

Die **gemeldete Person** enthält eine Referenz zu einer teilnehmende Person, die der Bestellung hinzugefügt werden soll und eine Anzahl. Teilnehmende Person

ist dabei die Verwaltungsgröße, während gemeldete Person ein Intermediate Object[5] (Zwischenobjekt) ist, das zu einer Bestellung gehört und sich dadurch auszeichnet, welche Anzahl von welcher teilnehmenden Person für eine Bestellung gemeldet wurde.

Mit **bestellten Kostformen** und **Kostformen** verhält es sich äquivalent. Kostformen haben eine Bezeichnung, einen Preis und eine Einheit. Außerdem kann jede Kostform in genau eine Kostformkategorie eingeordnet werden. Damit ist eine übersichtliche und themenverwandte Präsentation möglich.

Eine **Einheit** hat eine Bezeichnung.

Die Umsetzung dieses UML-Modells in ein Klassendiagramm befindet sich in 4.1.1 Modellierung.

2.4 Wireframes

Der Begriff Wireframe (Drahtmodell) wird größtenteils in der Webentwicklung verwendet und steht für die Darstellung des strukturellen Aufbaus einer Website. Es handelt sich, im Gegensatz zu einem Mock-Up aus 4.3.2 Entwurf der GUI mit Mock-Up, um rein grafische Darstellungen. Dabei wird wenig Wert auf Design, sondern eher auf die Existenz und grobe Anordnung der Elemente auf der Seite gelegt. Somit stehen Wireframes am Anfang des Designprozesses der grafischen Nutzerschnittstelle (GUI). Sie sind dafür gedacht, dem Designer eine Grundstruktur vorzugeben, ohne ihn in seiner kreativen Arbeit einzuschränken. Man kann für eine Website verschiedene Bereiche festlegen, die dann im Layout immer eingehalten werden. Das trägt zu einer hohen Übersichtlichkeit bei. Auch wenn Seiten verschiedene Layouts bekommen sollen, kann man mittels Wireframes die Unterschiede übersichtlich darstellen und eventuelle Vor- und Nachteile besser erörtern. Typisch für Webprojekte festgelegte Bereiche sind

5 Ein Intermediate Object besteht aus (mindestens) zwei Verknüpfungen zu jeweils einem Domain Object und zusätzlichen Daten, die die Verbindung zwischen den beiden Objekten spezifiziert.

- Header, meist mit einem Stimmungsbild
- Navigation
- Seiteninhalt
- Footer

Der Bestellservice soll für den eingeloggten Benutzer möglichst einfach zu bedienen sein. Es wird eine übersichtliche Grundstruktur benötigt. Da die unterschiedlichen Nutzergruppen auch verschiedene Ansichten und Bedürfnisse haben, muss man sich überlegen, ob jeder sein eigenes Grundlayout bekommen soll, oder alle das gleiche nutzen. Meist ist es nicht sinnvoll, völlig verschiedene Layouts für unterschiedliche Gruppen zu verwenden. Ein Trennung kann sinnvoll sein, wenn extrem unterschiedliche Anforderungen gestellt werden wie zum Beispiel der Unterschied zwischen normalem Nutzungsbereich und dem Administrationsbereich. Mit Administrationsbereich meine ich hier ein Backend, in dem man neue Nutzer anlegen, Nutzerdaten verwalten kann und beispielsweise eine Übersicht über die Aktivitäten der Nutzer bekommt. Im Fall des Bestellservices ist ein solches Backend nicht nötig. Das Anlegen der Nutzer und Daten erfolgt später im TYPO3-Backend. Daher muss es bei der Planung nicht mit beachtet werden. [jak] (S. 259)

3 Realisierungsansätze - Technologien und Werkzeuge

Die bisherige Projektbeschreibung und Modellierung sind auf noch nicht auf irgendeine Technologie zugeschnitten. Das hat den Vorteil, dass es sich dabei um eine übersichtliche Darstellung des Projektes handelt, die keine Technologie-spezifischen Details enthält. So ist die Darstellung auch für Fachfremde leichter zu verstehen und dient den am Projekt Beteiligten, aber auch den eventuell Dazustoßenden als Einführung in das Projekt.

Durch die Anforderung, dass der Bestellservice mit TYPO3 umgesetzt werden soll, werden jedoch schon einige Technologien festgelegt. Ihre Einsetzbarkeit für den Bestellservice wird dabei vorausgesetzt.

Was Extbase genau ist, werde ich im Kapitel *3.3.2 Extensionentwicklung mit Extbase* erläutern. Extbase allein ist aber nur ein Teil der Entwicklungsumgebung. Da TYPO3 in PHP (Hypertext Preprocessor) geschrieben ist und auf einem Webserver läuft, werde ich Aufbau und Funktionsweise eines solchen Webservers beschreiben. Zudem ermöglicht Extbase eine Implementierung der Software nach dem Domain-driven Design (DDD). Deswegen werde ich im folgenden Abschnitt 3.1 Domain-driven Design darauf eingehen, was hinter diesem Begriff steckt.

3.1 Domain-driven Design

Bei der Entwicklung von Software werden meistens vorhandene Systeme auf Computerebene nachgebildet. In der vorliegenden Diplomarbeit geht es zum Beispiel um die Nachbildung eines Bestellprozesses. Dieser Prozess wurde bisher schon mittels Bestellformular und Fax durchgeführt, soll aber nun mithilfe der

Computertechnik komfortabler werden. Zudem eröffnen sich mit der Digitalisierung des Vorganges auch neue Möglichkeiten, z.B. Statistiken dynamisch zu generieren.

Wenn sich der Kunde also dazu entscheidet seinen Geschäftsprozess mit Software nachbilden zu lassen, beginnt der Austausch zwischen ihm und dem Entwickler (siehe *3.1.1 Knowledge Crunching*). Der Entwickler erstellt im Kundengespräch ein Modell. Je weiter diese Gespräche voranschreiten umso wahrscheinlicher tritt bei unerfahrenen Entwicklern folgender Fall auf: Das Modell ist komplex und unübersichtlich geworden, weil sich viele Aspekte während des Gesprächs mehr oder weniger stark geändert haben und die Anpassungen schnell nachgetragen wurden. Dadurch haben sich viele unnötige Abhängigkeiten eingeschlichen. Das Modell ist dadurch unflexibel geworden und muss neu aufgesetzt werden. Das daraufhin erstellte Modell ist zwar etwas sauberer definiert, wird aber in weiteren Kundengesprächen bald wieder von Grund auf zu erneuern sein.

Eric Evans' "Domain-Driven Design" [eeddd] beschäftigt sich unter anderem mit diesem Problem. Sein Ziel ist es, dass der Entwickler selbst komplexe Systeme übersichtlich modelliert, sodass eine flexible Software entsteht. Das heißt auch, dass sie gut wartbar, anpassbar und auch erweiterbar ist, ohne dass die Übersichtlichkeit darunter leidet. Evans hat, bevor er das Buch schrieb, über Jahre hinweg diesen Ansatz erprobt und verfeinert. Sein Konzept findet in weiten Teilen der Programmiererwelt, unter anderem auch beim TYPO3 Entwicklerteam, großen Anklang.

Was Domain-driven Design genau ist, kann man schlecht sagen. Selbst der Geistige Vater führt in seinem Buch keine genaue Definition an, sondern bezeichnet es in erster Linie als eine "Philosophie", die auf der Basis der "objektorientierten Entwicklung" beschreibt, wie die "kritischen Aufgaben der Modellierung und des Designs von [domains][6]" bewältigt werden können [eeddd] (S. xix). Obwohl

6 Die Domain (Domäne) ist der mit der Software abzubildende Bereich der Realität

Evans viele praktische Erlebnisberichte anführt, ist DDD mehr als nur eine Anleitung, wie man Software modelliert. Es will die eigene Sichtweise auf das Projekt beeinflussen, aber auch den Umgang mit dem Kunden und mit Teammitgliedern - eben eine Philosophie.

In den folgenden Unterabschnitten will ich auf die Grundbausteine, die das DDD benutzt, eingehen.

3.1.1 Knowledge Crunching

Die Konzeptions- und Planungsphase eines jeden Softwareentwicklungsprojekts basiert auf intensiver Kommunikation zwischen dem Entwickler und dem Kunden. Beide Parteien gehen mit unterschiedlichen Voraussetzungen in diese Projektphase.

Der Kunde kennt die realen Abläufe, die mit der Software technisch abgebildet werden sollen, sehr genau. Er kann viel und ausschweifend darüber erzählen und wird dabei gelegentlich Fachbegriffe benutzen. Weil für ihn viele Einzelheiten und Abläufe zur unbewussten Routine geworden sind, setzt er manche Details als gegeben voraus und der Entwickler wird sie nie erfahren, wenn er nicht gezielt danach fragt.

Der Entwickler hingegen denkt in abstrahierten Modellen. Er versucht im Gespräch ein Modell zu erstellen, das die benötigten Anforderungen hinreichend abbilden kann. Auch er benutzt ganz automatisch im Gespräch Fachbegriffe aus seiner Programmierwelt. Er muss die Informationen zerlegen und die Einzelteile so strukturieren, dass ein um die unwesentlichen Bestandteile reduziertes Modell entsteht. Diesen Prozess bezeichnet Evans als *Knowledge Crunching* (Wissen zermalmen). Wenn während des Austauschs über das nachzubildende System neue Aspekte auftauchen, ist es unter Umständen nötig das Modell umzustrukturieren. Dank des Modellierungsansatzes nach dem DDD sollte das aber kein Problem sein. [t3mddd]

3.1.2 Ubiquitous Language

Wie in vielen Bereichen des täglichen Lebens ist auch in Teamprojekten jedweder Art die Kommunikation ein Grundpfeiler, mit dem das Projekt steht und fällt. Wie oben beschrieben verwenden sowohl Kunden als auch Entwickler ihre eigenen Fachbegriffe. Evans empfiehlt eine domänenspezifische Sprache – die *Ubiquitous Language* – zu entwickeln. Gewissermaßen eine Fremdsprache für Entwickler und Kunden, die beide erst lernen müssen, mit deren Hilfe die Domäne aber ideal und unmissverständlich beschrieben wird. Er geht so weit, dass diese Sprache nicht nur in der verbalen Kommunikation, sondern auch im Code und in den Dokumentationen eingesetzt werden soll. Dadurch soll es versierten Kunden auch möglich sein, den Quelltext wie ein Buch lesen zu können. Dafür braucht der Entwickler Disziplin, da man doch gelegentlich dazu neigt, kryptische Abkürzungen für Variablen oder Nichts sagende Methodennamen einzuführen, weil es gerade wieder einmal schnell gehen muss.

Dem Vokabular der Ubiquitous Language müssen unter Umständen neue Wörter hinzugefügt werden, wenn das Modell um neue Aspekte erweitert wird.

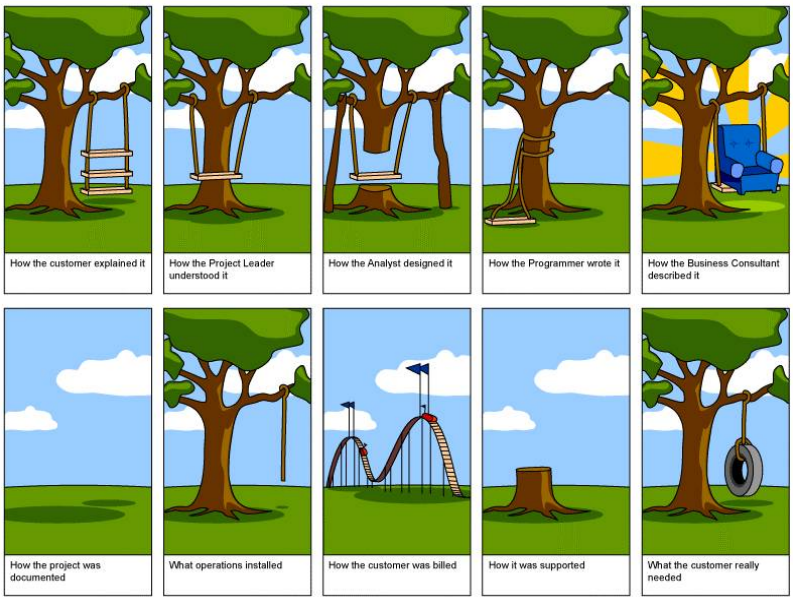


Abb. 3-1: Ein gewöhnliches Webprojekt [woul]

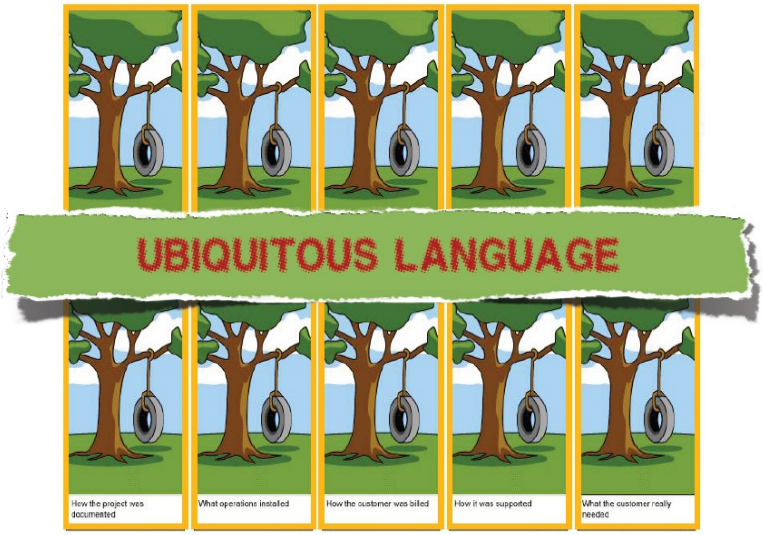


Abb. 3-2: Ein Webprojekt mit Ubiquitous Language [slide] (S. 73)

Das Ziel der Ubiquitous Language ist die Vermeidung von Missverständnissen, die sowohl dem Auftraggeber als auch dem Entwicklerteam schaden und einen reibungslosen Projektablauf verhindern.

In einem Blogeintrag von AOE-Media, einer renommierten Agentur, die sich selbst „The TYPO3 Company“ nennt, ist folgende Zusammenfassung zu lesen:

> *"Der wichtigste Vorteil von domain-driven Design liegt in der anforderungsorientierten Darstellung der Geschäfts- und Fachlogik (Domänenmodell) in einer nutzerorientierten Common Language (domänenspezifische Sprache). Das Ergebnis sind schnellere Analyse-, Design- und Entwicklungsprozesse und vor allem bedarfsgerechtere Softwarelösungen. Diese Lösungen bilden die Domäne des Kunden adäquat ab, sind änderungsfreundlich und lassen sich gut weiterentwickeln."*
>
> *- [aoeddd]*

3.1.3 Building blocks

Eric Evans lässt uns nicht ganz allein mit seiner Philosophie. Er zeigt ein paar *Building Blocks* (Bausteine) auf, mit denen man eine Domain modellieren und verwalten kann. Ein paar dieser Bausteine möchte ich hier näher betrachten.

Domain Models - Entities und Value Objects

Bei Domain Models handelt es sich um die Objekte, die den Abstraktionsprozess überstanden haben und somit in der Domain eine Rolle spielen. Objekte mit ihren Eigenschaften und Methoden, die durch die Prozesse in der Domain erstellt, verändert und wieder gelöscht werden können. *Entities* (Entitäten) sind dabei Objekte, die eine Identität besitzen und nicht allein durch ihre Merkmale bestimmt werden. Z.B. ist eine Person mit braunen Haaren nicht identisch mit einer anderen Person mit braunen Haaren. Sie wird also durch mehr ausgezeichnet, als nur durch die Haarfarbe. *Value Objects* (Wertobjekte) andererseits stellen Werte dar, die allein durch ihre Eigenschaften identifizierbar sind. Farben können beispielsweise als Value Objects umgesetzt werden. Wenn man Rot mit Gelb mischt, hat man kein Rot mehr. Mit dem Orangeton ist eine neue Farbe entstanden. Während Entities in der Domain einen Lebenszyklus haben, sind Value Objects zustandslos.

Beim Modellieren der Domain ist die Entscheidung, ob ein Objekt eher als Entity oder Value Object implementiert werden sollte, manchmal gar nicht so einfach. In solchen Fällen macht man es davon abhängig, welche funktionale Rolle das Objekt in der Domain spielt. Und wenn man sich einmal falsch entscheidet ist das kein Beinbruch und kann nachträglich korrigiert werden. [t3mddd]

Domain Services

Services (Dienste) sind Klassen im Domain Model, die über den Tellerrand eines einzelnen Models hinaus schauen können. Sie erledigen Aufgaben, die ein Do-

main Model in seiner natürlichen Arbeitsweise allein nicht erledigen kann. Beispielsweise den Vergleich von mehreren Domain Models.

Repositorys

Ein *Repository* (Repositorium) repräsentiert alle Domain Models eines bestimmten Typs. Es ist eine Sammlung von Objekten, die gezielt nach bestimmten Kriterien abgefragt werden können. Das Repositorium ist dabei die einfache Schnittstelle zwischen der Domain und der im Hintergrund liegenden Infrastruktur. Objekte können hinzugefügt und entfernt werden. Für die Anwendung bleibt dabei verborgen, wo die Objekte hergeholt werden, die sie vom Repositorium abfragt. Diese Abstraktion macht es möglich, dass die Datenquelle im Hintergrund z.B. von dem Dateisystem zu einer Datenbank geändert werden kann, ohne dass man eine Programmzeile im Domain Model ändern müsste. Weitere Vorteile von Repositorien sind:

- Sie sind ein einfaches aber wirkungsvolles Modell für das Verwalten und Speichern von Objekten
- Sie entkoppeln Anwendung und Domain von der Persistierung (Speicherung) der Daten
- Sie lenken den Blick des Entwicklers weg vom Zugriff auf Objekte hin zu den Designfragen der Domain
- Sie erlauben ein einfaches Ersetzen des Datenmodells mit einer Dummy-Implementierung beispielsweise für Testzwecke

[t3n]

[eeddd] (S. 147 ff.)

3.2 Webserver und Co.

3.2.1 Webserver mit PHP

Der Einsatz einer Webserver-Client-Technologie hält die Systemanforderungen sehr niedrig. Der Nutzer braucht nur einen internetfähigen Rechner mit einem installierten Webbrowser (z.B. Mozilla Firefox, Internet Explorer). Als Webserver reicht ein moderner Server mit installiertem PHP. Weil ich TYPO3 Version 4.3 einsetze, wird PHP ab Version 5.2 vorausgesetzt. Zudem braucht der Webserver Zugriff auf eine relationale MySQL-Datenbank, die ich auch zum Speichern der Daten des Bestellservice nutzen werde. [t3sr]

PHP ist eine Programmiersprache, die auf dem Webserver benutzt wird, um angeforderte Webseiten dynamisch, zum Zeitpunkt des Anforderns, zusammenzubauen. Dadurch können Daten vom Nutzer, beispielsweise aus Formularen, mit in die Seite eingebaut werden.

So wird ein Web-Request verarbeitet:

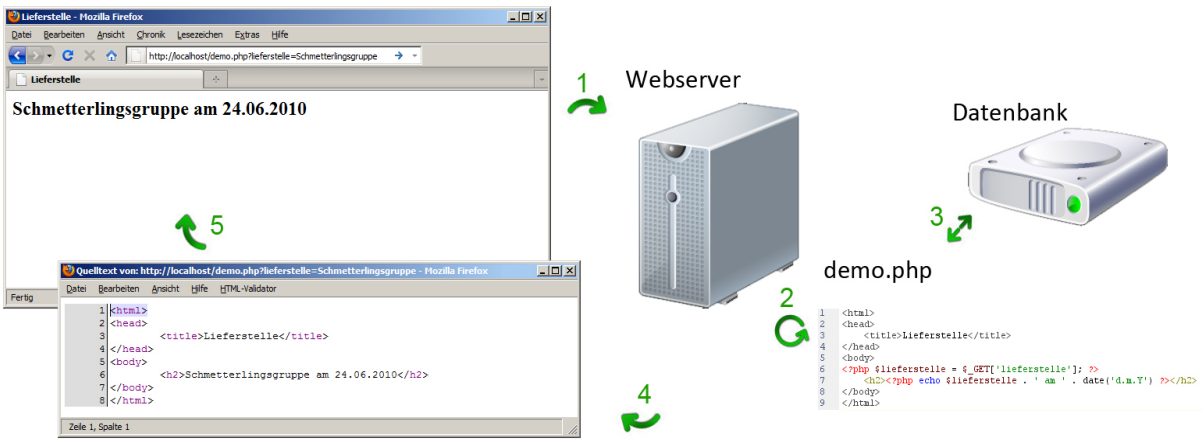


Abb. 3-3: Ein Webrequest wird verarbeitet

1. Der Web-Request wird zum Server gesendet

2. Die PHP-Engine löst die Codeteile aus der PHP-Datei auf

3. Ein Datenbankzugriff ist dabei optional

4. Das fertige HTML-Dokument wird zum Browser zurückgeschickt

5. Der Browser stellt die Antwort des Servers dar

PHP stellt dabei eine mächtige Sammlung von Funktionen bereit, mit denen sich Datenbank- und Dateisystemzugriffe und allerlei Berechnungen durchführen lassen. Außerdem bietet PHP seit Version 3 ansatzweise Möglichkeiten der objektorientierten Programmierung. Diese wurde bis Version 4 allerdings eher stiefmütterlich behandelt. Erst mit PHP Version 5 [phpoo] wurde eine an Java erinnernde Objektorientierung eingeführt. Die objektorientierte Programmierung ist eine wichtige Voraussetzung für Domain-driven Design und wird von Extbase mit dem Domain Model konsequent gefordert. Unter anderem deswegen nutzt Extbase auch PHP ab Version 5.2. Ein wichtiges Stichwort im objektorientierten Einsatz von PHP ist *Vererbung*. Man kann PHP-Klassen voneinander erben lassen. Damit übernehmen die *Kinder* die Eigenschaften und Methoden der *Eltern* und spezialisieren diese.

3.2.2 Die Datenbank

In einer Datenbank, wie MySQL, können Daten sytematisch gespeichert und ausgelesen werden. Eine Datenbank besteht aus beliebig vielen Tabellen, in denen jeweils Datensätze abgelegt werden. Dafür werden für jede Tabelle Felder definiert, die mit einzelnen Daten gefüllt werden können. Jedes Feld hat einen bestimmten Datentyp, z.B. TEXT (Zeichenkette) oder INT (Ganzzahl), aber auch DATETIME für Datumsangaben und einige mehr.

Datenbank

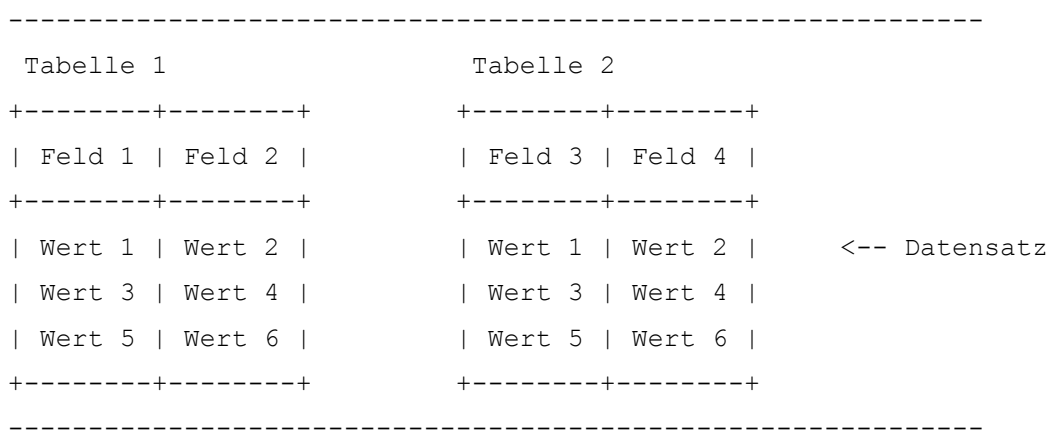


Abb. 3-4: Beispiel einer Datenbank

Für den Zugriff auf relationale Datenbanken gibt es spezielle Sprachen. Die am weitesten verbreitete ist die Structured Query Language (SQL), die auch im vorliegenden Projekt verwendet wird. Die Schlüsselwörter sind in englischer Sprache verfasst. Der Aufbau einer Abfrage (Query) ist auch an die englische Syntax angelehnt. Eine einfache Beispielabfrage sieht wie folgt aus, wenn sie alle (*) Datensätze der Tabelle "lieferstellen" liefern soll, deren zugeordneter "kunde" die ID 5 hat.

```
SELECT * FROM `lieferstellen` WHERE `kunde` = 5;
```

Im Bestellservice werde ich nicht viel mit der Datenbank direkt zu tun haben, da Extbase eine eigene Abstraktionsschicht für die Datenbank bereitstellt. Ich muss also nicht die Query selbst schreiben, sondern die Schnittstelle mit den Daten füttern, die Extbase braucht, um eine SQL-Query daraus zu bauen.

3.3 Frameworks und Bibliotheken

Mit dem oben beschriebenen PHP-fähigen Webserver könnte man die Aufgabenstellung eigentlich schon umsetzen. Allerdings wäre es nicht sinnvoll, da alle Funktionalität von Grund auf implementiert werden müsste. Anmeldung und Verwaltung der Nutzer, Datensatzpflege, Datenbankzugriffe und vieles mehr. Außer-

dem spielen die Sicherheit und die Stabilität des Systems eine große Rolle. Um diese zu gewährleisten bedarf es schier unendlicher Testphasen und enormen Entwicklungsaufwandes. Außerdem ist es äußerst ineffizient, Probleme zu lösen, die andere schon längst gelöst haben. Besonders wenn man freie Verfügung über die Arbeit der anderen hat. Was ich damit sagen will: Man muss das Rad nicht neu erfinden, sondern kann auf hochwertigen, verfügbaren Code zurückgreifen. Ob man den Code selbst verfasst hat, oder dieser im Rahmen eines Open Source Projektes entstanden ist, ist bei gleicher Funktionalität nur zweitrangig. Darüber hinaus gibt es weiter Vorteile einer Entwicklergemeinschaft, die nicht unterschätzt werden sollten. Die Möglichkeit der kostenfreien Nutzung erfreut den Auftraggeber, die Qualität des Codes erfreut den Entwickler. Das waren für mich die ausschlaggebenden Beweggründe, den Bestellservice komplett auf Basis von Open Source-Projekten zu realisieren. In den folgenden Unterabschnitten werde ich auf das Content Management System (CMS) TYPO3 eingehen, besonders dabei die Extensionentwicklung mit Extbase und der neuen Template Engine Fluid betrachten und erläutern. Da ich auch JavaScript eingebracht habe, werde ich auf die genutzte jQuery-Bibliothek eingehen.

3.3.1 Das TYPO3 Content Management System

TYPO3 - Die Überschrift verrät es schon - ist ein Content Management System (CMS). Da es schon viel gedruckte Literatur über TYPO3 gibt und für den Bestellservice auch nur ein kleiner Teil des ausgedehnten Funktionsumfangs von TYPO3 benötigt wird, beschränke ich mich bei der Beschreibung auf die für den Bestellservice wesentlichen Teile.

In TYPO3 existiert ein Backend, mit dem die Website, und alles was dazu gehört, verwaltet werden kann. Der Bestellservice wird keine eigene Verwaltung für die Grundstruktur der Daten bekommen, sondern die übersichtlichen Backend-Formulare von TYPO3 nutzen.

TYPO3 ist sehr gut erweiterbar. Neben den voreingestellten Inhaltselementen, wie z.B. Text, Bild und Tabelle, können auch eigene, so genannte Extensions, erstellt und als Plugin in die Seite eingefügt werden. Der Bestellservice wird als eine solche Extension entwickelt. Damit die Daten der Extension von TYPO3 verwaltet werden können, gibt es Konfigurationsskripte wie ext_localconf oder ext_tables, in denen TYPO3 die nötigen Informationen über die Extension mitgeteilt werden.

TYPO3 hat eine flexible Konfigurationssprache – TypoScript. Mithilfe von TypoScript können Extensions flexibler und für verschiedene Bedürfnisse anpassbar gemacht werden. Extbase wird vermehrt Gebrauch von TypoScript machen um die Framework-Konfiguration einzurichten. Außerdem können viele Aspekte, die Extbase per Konvention vorgibt mit TypoScript umkonfiguriert werden.

3.3.2 Extensionentwicklung mit Extbase

pibase - Der Stand der Technik vor Extbase

Eine wichtige Stärke von TYPO3 ist seine Erweiterbarkeit durch spezifische Extensions, auch Plugins genannt. Obwohl das eigene Plugin in eine PHP-Klasse eingeschlossen wird, möchte ich die Programmierung von Extensions nach dem klassischen Prinzip nicht als objektorientiert bezeichnen, da die Klassen eher genutzt werden um Namensräume zu definieren und die Modularität im System zu wahren. Die Basisklasse *tslib_pibase*, von der die eigene Extensionklasse erbt, stellt dem Entwickler ein mächtiges Set von Funktionen zur Verfügung. Von Datenbankfunktionen über Lokalisierung, Sortierung und Filterung von Datensätzen bis hin zur Möglichkeiten der Linkgenerierung und der visuellen Ausgabe. Der aufmerksame, versierte Leser erkennt schon hieraus, dass vom Standpunkt des MVC[7] betrachtet, hier alle Komponenten in einer Klasse gebündelt werden. Das bedeutet unter anderem, dass die Wartbarkeit des Codes verkompliziert wird.

7 Model-View-Controller: Architekturmuster in der objektorientierten Programmierung, das die Trennung der drei Komponenten Model, View und Controller vorsieht.

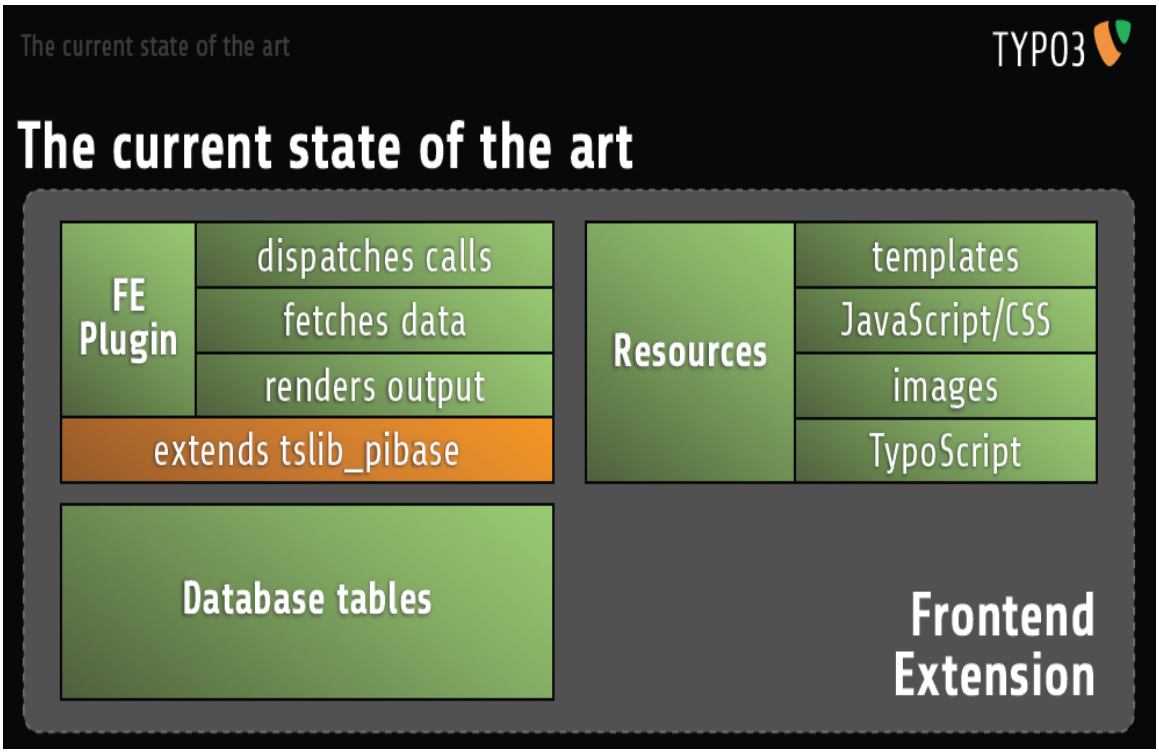


Abb. 3-5: Der Stand vor Extbase [slide] (S. 6)

Der Entwickler kümmert sich um die Anfrage des Nutzers, das Sammeln der angefragten Daten, unter Umständen auch das Schreiben und Lesen in der Datenbank und um die Ausgabe der Daten. Dazu werden Ressourcen wie Templates (siehe *3.3.3 Fluid*), Bilder und JavaScript eingebunden.

Der ungeordnete Aufbruch

Den Missstand, dass die komplette Funktionalität der Erweiterung in einer einzigen Klasse gebündelt wird und man sich um jeden Bestandteil selbst kümmern muss, haben in der Vergangenheit schon viele TYPO3-Entwickler gesehen, und mit eigenen Lösungen mehr oder weniger behoben[8]. Der Nachteil an den individuellen Initiativen ist: Es gibt unzählige verschiedene Basis-Extensions, die beispielsweise MVC ermöglichen. Damit werden Abhängigkeiten zwischen Extensions geschaffen. Um die Extension *pt_list* erfolgreich zu installieren sind zum Beispiel die MVC-Extension *pt_mvc* und die Bibliothek *pt_tools* notwendig. Jede

8 Quellen: rn_base, lib/div, pt_mvc, mvc, oelib

dieser Basisextensions wird nur von einer kleinen Gruppe von Entwicklern genutzt. Deshalb kann es unter Umständen nötig werden, dass bei der Installation dreier Extensions verschiedener Hersteller zusätzlich noch mindestens drei weitere Basisextensions hinzugefügt werden müssen.

Extbase – die offizielle Lösung

An dieser Entwicklung erkennt man das Streben der Entwickler nach objektorientierten Konzepten, die aber bisher noch nie solide von offizieller TYPO3-Seite unterstützt wurden. Das wird nun anders. Das MVC-Framework "Extbase" hat zwei Ziele: Die eben beschriebene Lücke eines offiziellen MVC-Frameworks zu schließen, und den Weg zum neuen FLOW3-Framework[9] [f3org] zu ebnen.

Streng nach dem Ansatz des DDD lenkt Extbase den Blick des Entwicklers auf das zu entwickelnde Modell und übernimmt selbst große Teile der Persistierung, Validierung und Filterung der Daten. So muss sich der Entwickler nicht mehr um Datenbankzugriffe kümmern, sondern kann den Fokus ganz auf die Probleme legen, die er mit seiner Extension lösen will.

Damit erleichtert Extbase das Entwickeln von Extensions, denn gerade bei Plugins mit mehreren zu verwaltenden Datentypen verbringt man sonst viel Zeit mit Create, Update, und Delete-Konstrukten. Bei Extbase wird das schon 'Frei Haus' mitgeliefert. [slide][extpod]

> *"So the framework does the boring stuff and we do the really cool interesting."*
>
> *- Sebastian Kurfürst, TYPO3 Core Developer*

Erstmals als stabil veröffentlicht wurde Extbase im November 2009. Seitdem wurden aber so viele verbessernde Veränderungen vorgenommen, dass ich es

9 FLOW3 ist ein eigenständiges Framework, welches das Entwurfsmuster des DDD voll unterstützt. Extbase basiert zum großen Teil auf dem Code von FLOW3 und bildet die Brücke zwischen TYPO3 Version 4 und der FLOW3 basierten TYPO3 Version 5

vorziehen musste mit einer sich ständig verändernden Grundlage zu arbeiten. Dafür habe ich aktuelle Änderungen in Extbase (und Fluid) mittels SVN[10] direkt in meinem Entwicklungssystem integriert.

3.3.3 Fluid

Bei statischen Webseiten werden vom Client HTML-Seiten vom Server angefordert, die genau so zum Client geliefert werden, wie sie sind. Für dynamische Webseiten ist das aber nicht möglich. Es muss beispielsweise an der Stelle, wo das aktuelle Datum stehen soll, ein Platzhalter eingefügt werden. Ein Dokument mit solchen Platzhaltern wird Template (Vorlage) genannt. Wie der Platzhalter aussieht und welche Funktionalität dahinter steht, ist von der Template Engine[11] abhängig. Diese nimmt das Template und ersetzt die Platzhalter mit dem aktuellen Inhalt.

Bisherige Ansätze für Templating

Im klassischen TYPO3-Templating werden Marker (Platzhalter) und Subparts (Unterabsätze) eingesetzt. Subparts sind gleichnamige Marker, die einen Bereich im Template einschließen. Das Template selbst enthält keine Kontrollstrukturen. Alles, was an Logik, besonders auch die Wiederholungen von Subparts, in die Ausgabe soll, muss in der eigenen Extension zusätzlich im PHP verarbeitet werden. Das bedeutet, dass alle Funktionalität, die im Template genutzt werden soll, in der eigenen Extension im PHP vorbereitet werden muss.

Template:

```
###LISTE###
<ul>
###LISTENELEMENT###
   <li>###TITEL###</li>
###LISTENELEMENT###
```

10 Mit SVN (Subversion) können Dateien versioniert werden, d.h. Änderungen an der Datei werden gespeichert. SVN ist sehr gut für eine Entwicklergemeinschaft geeignet.

11 Die Template Engine ist eine Software, die Templates verarbeitet und enthaltene Platzhalter ersetzt.

```
</ul>
###LISTE###
```

```
$subpart = getSubpart("LISTENELEMENT");
$output = '';
foreach($datensaetze as $datensatz){
   $output .= substituteMarker($subpart, 'TITEL', $datensatz['titel']);
}
```

Ein Nachteil an dieser Art des Templating ist, dass der Designer, wenn er sich um die Ausgabe der Extension kümmern soll, sowohl HTML als auch PHP beherrschen muss, denn komplexere Templates kommen ohne Kontrollstrukturen nicht aus.

Warum also jetzt ein neuer Ansatz?

Die Entwickler von Fluid haben auch andere, existierende Template Engines getestet und einiges davon abgeschaut, aber sie genügten jeweils nicht den Anforderungen. *PHPTAL*, ursprünglich in Python programmiert, aber in verschiedene anderer Sprachen eingebettet, nutzt zusätzliche HTML-Tag-Attribute, wodurch zwar wohlgeformtes XML entsteht, aber die HTML-Tags auch invalide Tag-Attribute bekommen. Eine wenig intuitive Semantik, die Möglichkeit PHP im Template zu verwenden und die schwere Erweiterbarkeit waren weitere Kriterien, nicht auf PHPTAL umzusatteln. [slidefluid] (S. 19 ff.)

Auch das PHP4 basierte *Smarty* bringt viel nützliche Funktionalität, aber auch ein paar Probleme mit sich. So gibt es zwar die Möglichkeit der Erweiterung der Funktionalität, jedoch keine eindeutige Namespace-Trennung, sodass sich benutzerdefinierte Funktionen gegenseitig überschreiben können. Außerdem ist Smarty durch seine PHP4-Basis nicht komplett objektorientiert programmiert worden. An dieser Stelle sei angemerkt, dass die Entwicklung für Fluid im Oktober 2008 offiziell begann, an das objektorientierte Smarty 3 war da also noch gar nicht zu denken. [slidefluid] (S. 17 f.)

Meiner Meinung nach ist der größte Vorteil von Fluid die Intuitivität, die den Fluid-Entwicklern auch von Anfang an sehr am Herzen lag. Wortwahl und Syntax müssen nicht unbedingt von PHP übernommen, sondern können frei gewählt werden, um Position und Funktion in der Domäne zu erfüllen. Damit werden auch die Anforderungen des Domain-driven Designs erfüllt.

"Einfachheit ist die ultimative Vollkommenheit"

- Leonardo DaVinci

Die erklärte Anforderung an Fluid ist es, eine einfache, elegante, sauber erweiterbare Template Engine zu sein, die den Templateautor unterstützt [slidefluid] (S. 27 ff.). Meine Erfahrungen bei der Arbeit mit Fluid bestätigen, dass diese Anforderungen eingehalten wurden. Eine extrem kurze Eingewöhnungszeit in die Syntax, die Möglichkeit, mit Objekten umzugehen und die Erweiterung mit View-Helpern zeugen von einem gelungenen Konzept für eine Template Engine.

An dieser Stelle einmal ein Beispiel, wie Ausgaben im Fluid-Template realisiert werden. Die Vorbedingung ist, dass der Template Engine (dem View) das PHP-Objekt "$foodform" (Kostform) übergeben wurde. Dieses Objekt besitzt eine Eigenschaft "$title" (Titel) und eine zugehörige Get-Methode getTitle(). Eine weitere Eigenschaft von "$foodform" ist "$unit" (Einheit), ebenfalls ein Objekt mit einer Eigenschaft "$title" und der Get-Methode[12].

Auf Objekte zugreifen (Object accessors)

Wie man mit Fluid sehr einfach auf Objekte und ihre Eigenschaften zugreifen kann, soll folgendes Beispiel verdeutlichen.

Beteiligt sind zwei miteinander verknüpfte Objekte. Eine Kostform (foodform) und ihre Einheit (unit). Verknüpft sind die beiden über das "unit"-Feld in der Ta-

12 Mit Get-Methoden kann man auf Objekteigenschaften zugreifen. Ein Objekt mit der Eigenschaft $name kann z.B. die Get-Methode *getName()* implementiert haben

belle "foodform", denn da steht die ID der Einheit aus der Tabelle "unit". Die Verbindung muss in TYPO3 auch noch konfiguriert werden, das wird für folgendes Beispiel als gegeben vorausgesetzt.

DB-Tabelle: "foodform"

```
+------+---------+--------+
|uid   | title   | unit   |
+------+---------+--------+
|1     | Zucker  | 2      |
+------+---------+--------+
```

DB-Tabelle: "unit"

```
+------+-----------+
|uid   | title     |
+------+-----------+
|1     | Portionen |
|2     | Päckchen  |
+------+-----------+
```

Ich möchte die Bezeichnung der Kostform und die Bezeichnung der zugehörigen Einheit ausgeben. Das Fluid-Template sieht so aus:

```
<p>
   Die Bezeichnung der Kostform lautet: {foodform.title},
   ihre Einheit ist {foodform.unit.title}
</p>
```

Die notwendigen Getter-Methoden werden automatisch aufgerufen. Im Beispiel :

```
<p>
   Die Bezeichnung der Kostform lautet: <?php echo $foodform->getTitle(); ?>,
ihre Einheit ist <?php echo $foodform->getUnit()->getTitle(); ?></p>
```

Als endgültige Ausgabe steht dann da:

```
<p>Die Bezeichnung der Kostform lautet: Zucker, ihre Einheit ist Päckchen</p>
```

Die Punktnotation funktioniert nicht nur bei Objekten, sondern auch bei Arrays. So kann man mit

```
{settings.teaser.action}
```

auf das Array

```
$settings['teaser']['action'] = 'showActualTeaser';
```

zugreifen und es wird "showActualTeaser" ausgegeben.

[slidefluid] (S. 37)

Parsen und Rendern

Das Template wird erst geparst und dann gerendert. Parsen bedeutet, dass die Template Engine das als Zeichenkette vorliegende Template scannt und verschiedene Textbereiche herausfindet. Es gibt normalen Text und Fluid-eigene Platzhalter mit besondere Bedeutung, wie zum Beispiel die bereits beschriebenen "Object accessors" oder ViewHelper-Tags, die weiter unten näher erklärt werden. Das Template liegt nach dem Parsen als Liste von Node-Objekten (Knoten) vor, z.B. "TextNode" oder "ObjectAccessorNode". Dann erfolgt der Aufruf zum Rendern des Templates. Der Renderer weiß, wie die unterschiedlichen Knoten behandelt werden müssen und ruft dann beispielsweise bei ObjectAccessorNodes die notwendigen Getter-Methoden auf. Das gerenderte Template ist wieder eine Zeichenkette, diesmal aber sind die Fluid-eigenen Platzhalter mit konkreten Werten ersetzt.

Passives und aktives Templating[13] - Fluid Performance

Das klassische, markerbasierte TYPO3-Templating funktioniert so, dass vom Programmierer explizit Marker bereitgestellt werden, die dann im Template genutzt werden können. Eine TYPO3-API-Funktion fügt dann Marker und Templa-

13 Passives und aktives Templating sind keine offiziellen Begriffe. Ich habe sie eingeführt, um den Unterschied der beiden Formen des Templating zu verdeutlichen.

te zusammen. Bei dieser Art des passiven Templates werden Marker bereitgestellt, die möglicherweise im Template gar nicht benutzt werden. Je nach Komplexität des Inhaltes der Marker (z.B. Berechnung der Fakultät eines Wertes, welcher der Addition und Subtraktion aus der Formel der Anzahl der Datensätze mehrerer Datenbanktabellen unterschiedlicher Datenbankserver entspricht. Dieser Wert ist ein Parameter eines Seitenlinks, der mittels TypoScript-Optionen manipuliert werden kann. Macht vielleicht nicht viel Sinn, dauert aber unglaublich lange) ist es möglich, dass zeitraubend Marker erstellt, aber nicht genutzt werden. Besonders unangenehm ist diese Art von Templating, wenn Caching nur bedingt möglich ist, z.B. bei Berechnungen im Zusammenhang mit der aktuellen Uhrzeit.

Der Vorteil des passiven Templating ist, dass das Template nicht erst geparst werden muss, also eine grundlegende Ressourcen- und Zeitersparnis stattfindet. Nachteilig ist aber, dass ein beträchtlicher Teil der Performance beim Füllen der nicht benötigten Marker verloren geht, egal ob diese gecacht sind oder nicht. Das ist besonders für komplexe Projekte ein dicker Minuspunkt.

Das Gute an der Methode des aktiven Templates ist, dass wirklich nur die Funktionen aufgerufen werden, die den Output auch wirklich beeinflussen. Aktiv wird das Template durch den Parser, welcher die Platzhalter aus dem Template nutzt. Das Parsing eines Templates verbraucht Ressourcen. Wenn das Template nicht verändert wird, ist das Parsing-Ergebnis aber immer dasselbe, egal ob die aufzurufenden Methoden beim Rendering dann komplex oder zeitabhängig sind. Das bedeutet, dass das Ergebnis des Parsing im Cache zwischengespeichert werden kann, bis das Template verändert wird. Statt zu parsen muss also nur noch der Cache ausgelesen werden, was unter Umständen einen deutlichen Performancegewinn ausmacht.

Das Beste an der Sache ist, dass sich darum weder Programmierer noch Templateautor kümmern müssen. Vom Programmierer muss lediglich sichergestellt werden, dass die Template Engine auf die durch den Templateautor angeforderten Ei-

genschaften und Methoden zugreifen kann. Je nach Möglichkeit kann innerhalb der angeforderten Methoden auch gecacht werden, was die Performance wiederum steigert.

Alles in allem ist für sehr einfache Templates das passive Templating das schnellere. Allerdings ist der programmiertechnische Aufwand ein größerer als beim aktiven Templating, wo die Template Engine große Teile der Funktionalität schon bereitstellt und eine Trennung von reinem Template und dahinter stehender Renderlogik ressourcenschonend möglich macht. Das macht das aktive Templating zum saubereren Ansatz und für komplexere Templates auch eindeutig zum Performance-Sieger.

Ein Mischform aus aktivem und passivem Templating ist, wenn das Template gezielt nach bestimmten Markern abgefragt wird, um sie im Falle der Existenz im Template zu rendern und ansonsten zu überspringen.

ViewHelpers

Der Object accessor ist einzig und allein für das Auslesen von Eigenschaften der an den View übergebenen Objekte da. Eine weitere in Fluid umgesetzte Philosophie ist, dass die Ausgabelogik (view logic) von der Geschäftslogik (business logic) getrennt wird. Und nicht nur das - die Ausgabelogik wird im Template integriert. Das hat mehrere Vorteile. Zum einen ist kein anderer Softwarebestandteil als der View dafür verantwortlich, wie die Daten präsentiert werden. Eine E-Mail-Adresse kann zum Beispiel als Text, codiert oder in Form eines Gravatars[14] ausgegeben werden. Für alle drei Möglichkeiten müsste der Controller die E-Mail-Adresse unterschiedlich codiert an den View übergeben. Aber was hat der Controller denn den View in seiner Arbeitsweise zu beeinflussen?

14 Ein Webdienst, der es möglich macht, an einer Stelle im Internet (gravatar.com) sein Bild hochzuladen, um es dann überall, wo man seine E-Mail-Adresse eingibt, zu sehen. Damit muss man nicht in jedem Blog ein Bild hochladen, sondern nur seine E-Mail-Adresse eingeben. Zur Sicherheit wird die E-Mail-Adresse MD5-codiert. Mehr dazu unter http://www.gravatar.com

Im Fluid-Kern selbst ist keine Ausgabelogik verankert. Jede Art von Ausgabelogik kann und muss über ViewHelper integriert werden. So gibt es im Fluid-Paket standardmäßig mitgelieferte ViewHelper, wie den CountViewHelper zum Ermitteln der Größe eines Arrays (siehe folgendes Beispiel), den LinkViewHelper, der TYPO3-Funktionalitäten zur Linkgenerierung nutzt, aber auch Kontrollstrukturen, wie If-, Then- und ElseViewHelper oder den ForViewHelper.

```
Gesamtzahl der Kostformen: <f:count subject="{foodforms}" />
```

Hinter jedem ViewHelper steht eine einzelne Klasse, die dessen Funktionalität realisiert. Hier exemplarisch der CountViewHelper aus dem Fluid-Paket:

```
Tx_Fluid_ViewHelpers_CountViewHelper extends
Tx_Fluid_Core_ViewHelper_AbstractViewHelper {

   /**
    * Counts the items of a given property.
    *
    * @param array $subject The array or ObjectStorage to iterated over
    * @return string The bumber of elements
    * @author Jochen Rau <jochen.rau@typoplanet.de>
    * @api
    */
   public function render($subject) {
      return count($subject);
   }
}
```

Der CountViewHelper ist denkbar einfach. Die Klasse erbt vom AbstractViewHelper, womit der korrekte Aufruf der Methode render() sichergestellt wird. Im Doc Comment[15] werden die erwarteten Parameter registriert. So wird hier beispielsweise die Variable namens "$subject" als array erwartet. Das wird mit der @params-annotation (Anmerkung) gefolgt von erwartetem Datentyp und Variablenname notiert. Fluid ordnet die zu übergebenden Parameter, falls mehrere vorhanden, und übergibt sie, wie sie von der render()-Methode erwartet werden. Fehlt ein Parameter oder er hat den falschen Datentyp, wird eine angepasste

15 Kommentarblock vor einem bestimmten Schlüsselwort im Quelltext

Fluid-Exception[16] geworfen. Somit übernimmt Fluid hier die Validierung der übergebenen Variablen. Auch eigene Validatoren können über die Annotations registriert werden.

Die Methode render() tut nichts weiter, als die PHP-Funktion count() aufzurufen und deren Rückgabe durchzuführen. Damit wird im Beispiel so etwas wie: "Gesamtzahl der Kostformen: 3" ausgegeben.

ViewHelper können neben der Tag-Schreibweise auch mit der sogenannten Inline Notation aufgerufen werden. Das Beispiel von oben sieht in der Inline Notation so aus:

```
{f:count(subject: foodforms)}
```

Die Inline Notation nähert sich optisch dem Aufruf einer Funktion an. Welche von beiden Schreibweisen man verwendet, hängt teils von der eigenen Vorliebe, teils vom Anwendungsfall ab. Beide haben ihre Stärken und Schwächen. Dabei sollte man die Übersichtlichkeit und die Semantik im Auge behalten. Manchmal ist die Tag-Schreibweise von ViewHelpern aber auch nicht möglich, zum Beispiel bei verschachtelten ViewHelpern:

```
<f:form.textbox name="amount" value="{f:count(subject: foodforms)}" />
```

Je nachdem, wie der ViewHelper aufgebaut ist, ist auch die Verkettung von ViewHelpern möglich:

```
{foodform.title
   -> f:format.padding(padLength: 20, $padString: '.')
   -> f:format.crop(maxCharacters: 15)}
```

Bei dieser Anweisung wird der Inhalt von `$foodform->getTitle()` (z.B. "Abendsuppe") an den Format_PaddingViewHelper[17] übergeben, welcher das Er-

16 Eine Exception (Ausnahme) wird meist dann erstellt, wenn im Programm ein Fehler auftritt, der die reibungslose Ausführung des Codes verhindert.

17 Beim Padding (Auffüllen) wird eine Zeichenkette mit einem vorgegebenen Zeichen auf eine bestimmte Zeichenanzahl ergänzt

gebnis (z.B. "Abendsuppe.....") an den Format_CropViewHelper[18] weiterleitet. Dieses Ergebnis wird dann ausgegeben (z.B. "Abendsuppe.....").

[slidefluid] (S. 38 ff.)

Kontrollstrukturen im Fluid-Template

Beispiel: Eine Reihe von Kostformen soll in einer HTML-Liste ausgegeben werden. Wenn aber keine Kostform gefunden wurde, soll ein Hinweistext erscheinen. Nach klassischem Markerprinzip war PHP-seitig immer eine Kontrollfunktion nötig, welche die Liste prüfte, entschied ob sie leer oder gefüllt ist und aus dem Template dann den entsprechenden Subpart auslesen musste.

```
###FOODFORM_LIST_SUBPART###
<ul>
   ###FOODFORM_LIST_ITEM###
   <li>###FOODFORM_LIST_ITEM_TITLE###</li>
   ###FOODFORM_LIST_ITEM###
</ul>
###FOODFORM_LIST_SUBPART###

###FOODFORM_LIST_EMPTY_SUBPART###
<p>Es wurden keine Kostformen gefunden</p>
###FOODFORM_LIST_EMPTY_SUBPART###
```

```
<?php
$foodformList = getFoodformList();
if (!empty($foodformList)){
   $subpart = getSubpart('FOODFORM_LIST_SUBPART');
   $output = renderSubpart($subpart, $foodformList); // in der Funktion werden
die Markerinhalte per Hand erstellt
}else{
   $subpart = getSubpart('FOODFORM_LIST_EMPTY_SUBPART');
   $output = $subpart; // keine Marker enthalten, nur plain text
}
?>
```

Mit Fluid sieht dieses Konstrukt folgendermaßen aus:

```
<f:if condition="{foodformList}">
   <f:then>
      <ul>
```

18 Beim Cropping (Abschneiden) wird die Zeichenkette auf eine vorgegebene Anzahl an Zeichen reduziert.

```
        <f:for each="{foodformList}" as="foodform">
          <li>{foodform.title}</li>
        </f:for>
      </ul>
   </f:then>
   <f:else>
      <p>Es wurden keine Kostformen gefunden</p>
   </f:else>
</f:if>

<?php
   // Controller / Action
   $foodformList = getFoodformList();
   $this->view->assign('foodformList', $foodformList);
?>
```

Das Template hat ähnlich viele Zeilen, die Ausgabelogik ist aber komplett vom Controller getrennt. Er entscheidet nur, welche Objekte dargestellt werden können, nicht ob überhaupt und in welcher Weise. Das fordert vom Templateautor natürlich die erweiterte Kenntnis über die Anwendung der existenten ViewHelper. Andererseits fordert das Templating mit dem Markerprinzip auch ein gewisses Maß an Vorstellungsvermögen, nicht wahr? Im Gegensatz zu anderen Template Engines ist die Schreibweise hier sehr intuitiv und daher auch für reine Designer nicht allzu schwer zu erlernen. [slidefluid] (S. 32 f.)

Formulare mit Fluid

Fluid stellt auch ViewHelper für Formulare zur Verfügung. Ein spezielles Feature gibt es für die Bearbeitung von Domain Models. Man kann dem Formular das zu bearbeitende Objekt übergeben. Dann gibt man für ein Formularfeld an, welche Eigenschaft des Objektes in dem Feld dargestellt werden soll. Fluid wird automatisch die Eigenschaft des Objektes und das Formularfeld zusammenbringen. Das gilt auch für das Bearbeiten eines Objektes, bei der Fluid den aktuellen Wert korrekt voreinträgt. Hier ein Beispiel für ein einfaches Fluid-Template mit Formular:

```
<f:form name="foodOrder" action="createOrUpdate" method="post"
object="{foodOrder}">
   <f:form.textbox property="orderer" />
   <f:submit value="Bestellernamen speichern" />
</f:form>
```

Hier wird dem Formular als Objekt eine Bestellung übergeben `object="{foodOrder}"`. Diese hat eine Eigenschaft namens Besteller `property="orderer"`, die eine Zeichenkette mit dem Namen des Bestellers repräsentiert. Wenn dem Formular eine Bestellung übergeben wird, nutzt Fluid den ObjectAccessor (siehe oben), um mit der Eigenschaft das Textfeld zu füllen. Durch Anwendung des ObjectAccessors ist es möglich, sich weiter durch die Objekte zu hangeln. Man stelle sich vor, "orderer" wäre ein Personenobjekt mit einem Adressobjekt "address", welches die Eigenschaft "street" hat. Die Textbox sähe dann folgendermaßen aus:

```
<f:textbox property="orderer.address.street" />
```

Validierung von Formularen

Da Formulare die Schnittstelle zum Nutzer sind, müssen sie auch entsprechend validiert werden. Das ist überhaupt meistens der aufwändigste Teil. Genau genommen gehört die Validierung von Formularen nicht zu Fluid, sondern wird von Extbase übernommen. Allerdings kommt es zu einem Wechselspiel zwischen Formular (Fluid) und Validierung (Extbase), da die Falscheingaben dem Nutzer meistens im Zusammenhang mit dem Formular präsentiert werden.

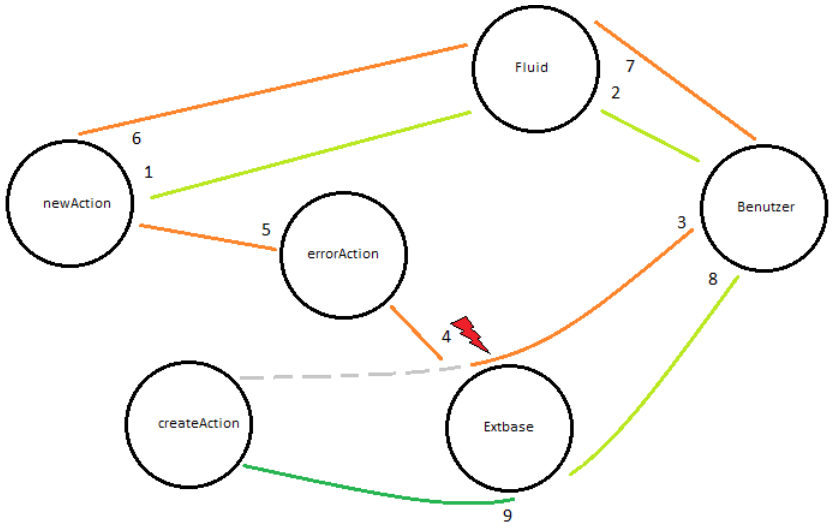


Abb. 3-6: Fluids Formular-Validierung

1. newAction ruft Fluid-Template auf
2. im Fluid-Template wird dem Nutzer ein Formular präsentiert
3. der Nutzer füllt das Formular mit invaliden Daten, ruft aber die createAction auf.
4. Extbase validiert, erkennt die falschen Nutzereingaben und ruft stattdessen die errorAction auf
5. errorAction bereitet die Fehler auf und leitet zur newAction zurück
6. newAction ruft wieder das Fluid-Template auf - diesmal wird das invalide Objekt übergeben, sodass Fluid die Formularfelder mit den vorher eingetragenen Daten vorausfüllen kann
7. Fluid präsentiert dem Nutzer das Formular inklusive der Fehlermeldungen
8. Nutzer korrigiert die Falscheingaben und ruft noch einmal die createAction auf
9. Extbase hängt sich dazwischen, validiert und weil es kein Fehl findet, wird createAction aufgerufen, in der das Objekt endgültig erzeugt wird.

In diesem Unterabschnitt habe ich gezeigt, wie Fluid die Nachteile oder Schwächen der anderen vorgestellten Template Engines kompensiert. Es ist komplett objektorientiert programmiert und kann mit Objekten und Arrays umgehen. Es kann auch außerhalb von HTML- und XML-Strukturen eingesetzt werden. Erweiterbarkeit gehört zum Grundkonzept von Fluid. Dabei werden Validierungen von Parametern direkt von der Template Engine übernommen, sodass der Code schlanker und sicherer wird. Auch für den Umgang mit Formularen ist Fluid gewappnet. Ich persönlich finde, dass ein klares Argument für Fluid dessen Intuitivität ist.

[slide], [extpod], [fluidpod]

3.3.4 jQuery – die JavaScript-Bibliothek

Was bisher geschah. Der Nutzer fordert eine Webseite an. Der Request gelangt zum Server, der daraufhin dynamisch die Rückgabe (Response) erstellt und zum Browser des Nutzers schickt (siehe *Abb. 3-3: Ein Webrequest wird verarbeitet S. 22*). Damit steht dem Nutzer nun eine Seite zur Verfügung, die zwar auf dem Server dynamisch generiert wird, ihm aber nun statisch vorliegt. Will er eine andere Seite sehen, klickt er einen Link, der einen weiteren Request auslöst, die Seite wird neu geladen und die neue Seite im Browser angezeigt. Die Dynamik auf Serverseite ist zwar schon ein großer Fortschritt zu statischem HTML, aber heutzutage erwartet der Nutzer mehr von der Website.

Die bisher genannten Technologien zum Generieren von Webseiten sind serverseitig. Das bedeutet, dass zur Laufzeit Inhalte dynamisch zusammengestellt werden. Was beim Nutzer ankommt ist aber im Grunde eine statische Webseite. JavaScript ist die bedeutendste Möglichkeit, solche Seiten dynamisch aufzupeppen. Dazu werden JavaScript-Funktionen geschrieben, die erst auf dem Rechner des Nutzers (clientseitig) ausgeführt werden. Bekannte Beispiele sind Bildergalerien oder Popup-Kalender. JavaScript wird dabei vom Browser interpretiert. Und genau das ist das Problem bei JavaScript und den andern clientseitigen Aspekten, die bei der Webprogrammierung zu bedenken sind. Unterschiedliche Browser und deren Versionen reagieren unterschiedlich, sodass der Entwickler zum Teil komplizierte Browserweichen[19] einbauen muss, um den Nutzern identische Ansichten zu geben oder Funktionalitäten auch in älteren Browsern zu simulieren.

Wie Serverseitig in PHP gibt es auch clientseitig Bibliotheken für JavaScript, die dem Entwickler die Arbeit erleichtern und immer wieder benötigte Funktionen möglichst kompakt bereitstellen. jQuery ist eine davon. Unter dem Motto "Write

19 Browserweichen sind Techniken, mit denen versucht wird, die Unterschiede in der Reaktion verschiedener Browser auf Quelltext zu kompensieren.

less, do more" (Schreibe weniger, tue mehr) angepriesen, unterstützt es den Entwickler massiv bei der Traversierung durch den DOM[20]-Baum, mit modernen, visuellen Effekten und bei AJAX-Requests[21].

Bestimmte Objekte des DOM-Baums auszuwählen und zu verändern ist einfach:

Gegeben sei folgender HTML-Ausschnitt:

```
<div id=”container”>
   <div id=”linkeSpalte”>
      <div>Inhalt der linken Spalte</div>
   </div>
   <div id=”rechteSpalte”>
      <div>Inhalt der rechten Spalte</div>
   </div>
</div>
```

Um das DOM-Objekt mit der `id=“rechteSpalte“` in die Variable `div` zu schreiben notiert man statt

```
div = document.getElementById("rechteSpalte");
```

mit jQuery nur noch

```
div = $("#rechteSpalte");
```

Die Syntax der so genannten Selektoren entspricht dabei genau der von CSS[22], sodass von folgendem Selektor alle div-Elemente gewählt werden, die sich in erster Ebene unter dem Element mit der `id="rechteSpalte"` befinden.

```
$("#rechteSpalte > div")
```

Ich habe jQuery für dieses Projekt gewählt, da es auch von anderen TYPO3-Community-Mitgliedern empfohlen wird. Zudem habe ich bisher zwar nur wenig, aber durchweg gute Erfahrungen mit jQuery in TYPO3-Frontend-Plugins ge-

20 DOM (Document Object Model): Repräsentation eines HTML- oder XML-Dokuments in Form eines Objektmodells, das die logische Struktur des Dokuments wiedergibt.[dom]

21 Ajax-Request: HTTP-Request, der mit JavaScript abgesetzt wird und mit dem es möglich ist, entfernte Daten anzufordern und in die Seite einzubauen, ohne dass die gesamte Seite neu aufgebaut werden muss.

22 Cascading Style Sheet: Definitionssprache für HTML- und XML-Elemente, mit der man die Darstellung von DOM-Elementen im Browser beeinflussen kann.

macht. Durch eine gute Dokumentation und konsistente API konnte ich bisher kleine Probleme meist schnell lösen.

Ich werde jQuery punktuell für die AJAX-Aufgaben verwenden. Aber auch der eine oder andere Effekt soll die Seiten des Projektes auflockern und vor allem intuitiver und auch für Laien leichter bedienbar machen. (Siehe 4.3.4 *jQuery und Ajax*)

4 Prototyperstellung - wesentliche Konzepte

In diesem Kapitel werde ich ausgewählte Lösungen zu den Problemen, die bei der Umsetzung des Bestellservices mit den eingesetzten Technologien aufgetreten sind, vorstellen. Die komplette Extension befindet sich auf einem Datenträger, der dieser Diplomarbeit beigelegt ist.

4.1 Das Domain Model

Nach dem Prinzip des DDD liegt der Fokus immer auf der Domain. Sie ist die Abstraktion der Realität. Dabei werden Aspekte, die für den nachzubildenden Geschäftsprozess nicht relevant sind, einfach weggelassen. Die Domain ist ein in sich geschlossenes Modell, das die interne Domainlogik beinhaltet. Das bedeutet, dass die Domain betreffende Aktionen nicht vom Controller gesteuert, sondern direkt von den Objekten der Domain durchgeführt werden. Wo diese Logik in der Domain genau liegt, hängt von der Art der Aktion ab. Zum Teil ist es schwierig zu entscheiden, wo genau welche Aktion implementiert wird. Das hängt dann oftmals von dem abzubildenden Geschäftsprozess ab. In den Mailinglisten wird dafür oft die Wendung "it feels natural" (es fühlt sich natürlich an) gebraucht. Dabei spürt man wieder den Philosophiecharakter des DDD. Man kann es nicht direkt erlernen, man muss es umsetzen, damit arbeiten und für sich selbst weiterentwickeln. Und genau diesen Prozess möchte ich mit der Diplomarbeit dokumentieren.

4.1.1 Modellierung

[23]Die objektorientierte Programmierung nutzt, wie der Name bereits andeutet, Objekte, um die Realität nachzubilden. Diese Objekte sind im Grunde sehr einfach aufgebaut. Sie haben bestimmte Eigenschaften und ein definiertes Verhalten. Dieses Verhalten unterscheidet ein Objekt auch von einer klassischen Variablen, die aus sich heraus nichts kann. Sie ist einfach nur da, repräsentiert einen Wert und kann verarbeitet werden.

Als traditioneller PHP-Entwickler ist man nach Kundengesprächen oftmals geneigt, die besprochenen Konzepte irgendwie in Datenbanktabellen zu packen. Man überlegt, welche Information in welches Feld gelegt werden kann, und wie das Feld am besten benannt wird. Domain-driven Design dagegen besteht darauf, den Fokus auf die Domain zu lenken und ihn dort zu belassen. Es soll nicht um statische Datenbankmodelle gehen, sondern um die an der Domain beteiligten Objekte und deren Verhalten innerhalb der Domain. Der Abstraktionsprozess der Domain soll nicht übertrieben technisch sein, sondern immer natürlich bleiben. Ansonsten entstehen unter Umständen generische Modelle, die zu abstrakte Namen haben, als dass sie von Unbeteiligten der Domain zugeordnet werden könnten. Das Prinzip der Ubiquitous Language spielt im Abstraktionsprozess eine wichtige Rolle und kann deshalb hier nicht oft genug genannt werden.

Leider kann man nicht ganz auf den Datenbankentwurf verzichten. Da Extbase von Haus aus die relationale Datenbank von TYPO3 unterstützt, ist es ratsam, auch die Daten der Extension in der Datenbank zu persistieren (nachhaltig speichern). Allerdings soll der Datenbankentwurf nicht der erste Schritt sein. Wenn man die Domäne richtig modelliert, entsteht der Datenbankentwurf (fast) ganz

23 Diesem Unterabschnitt liegt ein Vortrag von Jochen Rau und Sebastian Kurfürst zugrunde, der bei den T3DD09 (TYPO3 Developer Days 2009) gehalten wurde. Er ist teilweise nachzuvollziehen unter http://www.ustream.tv/recorded/1508956 [Stand: 25.06.2010]. Die Folien dazu: [slide]

von allein. Dafür konstruiert man aus dem Domänenmodell (siehe: *2.3 UML-Diagramm der Objekte*) nach den Extbase-Vorgaben ein Klassendiagramm[24]:

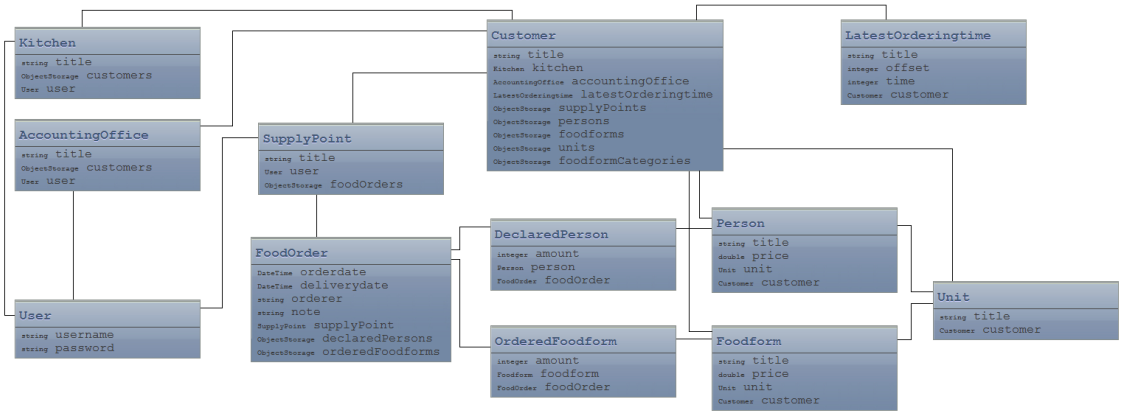


Abb. 4-1: Klassendiagramm der Objekte

In diesem Klassendiagramm fällt besonders auf, dass es in Englisch verfasst ist. Das hat folgende Gründe: Da TYPO3 ein internationales Projekt ist, das als Open-Source produziert wird, wird dringend empfohlen, dass Quelltext, Dokumentation und alles, was dazugehört, in englischer Sprache verfasst werden. Das ist besonders wichtig für Extensions, die im TYPO3-Extension-Repository (TER) veröffentlicht werden. Zudem ist es üblich, dass der Zugriff auf private Eigenschaften von Objekten mit sogenannten Getter- und Settermethoden erfolgt. Wenn man grundlegend in Englisch schreibt, kann man somit unschöne, 'denglische' Methodennamen wie getSpaetesteBestellzeit() vermeiden. Es ist also empfehlenswert, das Vokabular der Ubiquitous language in Englisch zu verfassen. Wenn die englischen Bezeichnungen dem Kunden nicht zugemutet werden können, muss die Ubiquitous language in zwei Sprachen existieren. Eine zusätzliche Aufgabe des Entwicklers ist dann, in beide Richtungen eindeutig zu übersetzen.

24 Klassendiagramme geben den Entwicklern eine Übersicht, welche Objekte in einer Domäne existieren und wie sie im Projekt miteinander in Verbindung stehen.

Aus dem Klassendiagramm lassen sich die Eigenschaften der Objekte gut ablesen. Daraus können dann die Datenbanktabellen zusammengestellt werden. Auch die Einträge im TCA können daraus erstellt werden. Extbase nutzt das TCA für das Mapping von Objekteigenschaften auf Datenbankfelder beim Speichern und auslesen von Objekten. Diese Arbeit ist zur Zeit leider noch notwendig. Im Bezug auf die Modellierung wird in naher Zukunft der neue Extbase-Kickstarter[25] vieles erleichtern. Leider befindet er sich derzeit noch in der Alpha-Entwicklungsphase.

Extbase stellt zwei (abstrakte) Basisklassen für Domain Models zur Verfügung. *Entity* und *Value Object*. Damit stützt es sich auf die Vorgaben des Domain-driven Design (siehe *3.1.3 Building blocks*). Man muss bei der Modellierung entscheiden, welche Rolle ein Objekt eines bestimmten Typs in der Domain spielt. Die Unterscheidung in Extbase, ob ein ValueObject oder eine Entity eingesetzt wird, bringt meines Erachtens zur Zeit programmtechnisch kaum einen Unterschied mit sich. Die Integration von Value Objects ist aber auch noch lange nicht perfekt und wurde erst nach der Veröffentlichung von Extbase 1.2, das am 22. Juni 2010 mit TYPO3 4.4 erschienen ist, verbessert werden.

4.1.2 Repositorien in Extbase

Repositorien werden genutzt, um auf die in der Datenbank gespeicherten Domain Models zuzugreifen. Sie sind Bestandteil des Domain-driven Designs, wie ich in *3.1.3 Building blocks* beschrieben habe. Der Grundaufbau von eigenen Repositorien in Extbase ist denkbar einfach:

```
01. class Tx_DitFood_Domain_Repository_FoodOrderRepository
```

25 Der Extbase-Kickstarter ist ein TYPO3-Backendmodul mit grafischer Oberfläche zur Modellierung von Domain Models für Extbase-basierte Extensions. Auf Basis des eingegebenen Modells generiert der Kickstarter auch die nötigen Dateien (u.a. Models, Konfigurationsdateien wie TCA und localconf, SQL-Datei). Damit ist er der Nachfolger des klassischen TYPO3-Kickstarters, der dabei hilft, pi_base-basierte Extensions zu modellieren.

```
extends Tx_Extbase_Persistence_Repository {}
```

Durch den Klassennamen *Tx_DitFood_Domain_Repository_FoodOrderRepository* weiß Extbase, dass es sich hierbei um ein Repositorium handelt, das sich um Objekte vom Typ *FoodOrder* kümmert. Dadurch kann es auch die zuständige Datenbanktabelle ermitteln. Und mit dieser Information ist der generelle Zugriff auf die Gesamtheit der FoodOrder-Objekte inklusive ihrer Daten schon gewährleistet. Darum stehen mit dieser einfachen Klassendefinition durch die Vererbung auch schon folgende Methoden zur Verfügung:

Für den schreibenden Zugriff:

- add($object)
- remove($object)
- removeAll()
- replace($existingObject, $newObject)
- update($modifiedObject)

Für den lesenden Zugriff:

- findAll()
- findByUid($uid)
- countAll()

Zudem sind noch sogenannte "magische Methoden" möglich. Ein FoodOrder-Objekt hat zum Beispiel die Eigenschaft *deliverydate* (Lieferdatum). Extbase kann aufgrund des Klassennamens auch die Eigenschaften der FoodOrder ermit-

teln. Dadurch ist, ohne eine Zeile zusätzlichen Quellcode, auch folgende Abfrage möglich.

- findByDeliverydate($deliverydate)
- findOneByDeliverydate($deliverydate)
- countByDeliverydate($deliverydate)

Während findBy...() ein Array mit allen Treffern findet, liefert findOneBy...() das erste dem Kriterium entsprechende Objekt zurück. Dass diesen Methoden als Kriterium auch Objekte übergeben werden können, ist im objektorientierten Extbase selbstverständlich. So ist bei der findByDeliverydate() als Parameter z.B. ein DateTime-Objekt möglich. Ein im Extbase-Kern befindlicher so genannter DataMapper ist für die Konvertierung der Objekte zuständig. In diesem Fall wandelt er das DateTimeObjekt in einen Unix-Zeitstempel[26]. [extblog]

Diese Funktionen sind für einige Aufgaben schon recht sinnvoll, kommen aber schnell an ihre Grenzen, z.B. wenn man nach mehreren Kriterien filtern möchte. Persistierung und Abfrage von Objekten erfolgt auf mehreren Ebenen. Von insgesamt sechs verschiedenen Persistenzebenen in Extbase schreibt Jochen Rau in seinem Blog. Hier kann man auch die genauen Funktionen der einzelnen Schichten nachlesen. [extblogpers]

- 1. Repository
- 2. Query
- 3. QueryObjectModel

26 Der Unix-Zeitstempel ist ein Ganzzahlwert, der die Anzahl der Sekunden seit dem 1.1.1970 angibt. Der 1. Juli 2010 08:00 Uhr (UTC) hat beispielsweise den Wert: 1277971200.

- 4. StorageBackend
- 5. DatabaseHandler
- 6. DataMapper

Der Vorteil der ersten beiden Schichten: Sie sind unabhängig von der Art, wie und wo die Daten gespeichert sind. Im QueryObjectModel hingegen werden schon datenbankspezifische Funktionen genutzt. Der Nachteil des hohen Abstraktionsgrades der ersten beiden Schichten liegt ebenso auf der Hand. Dadurch, dass nicht nur relationale Datenbanken mit der speziellen Datenbanksprache SQL angesprochen werden können, sondern auch das Dateisystem als Speicherort genutzt werden kann, gehen einige spezielle SQL-Funktionen durch die Abstraktion verloren. Besonders wenn man die bisherige Arbeit mit TYPO3 gewohnt ist, bei der es nicht unüblich war, eigene SQL-Querys zu schreiben, fehlen manche spezielle Funktionen. Für diesen Fall wurde das Schichtenmodell aufgeweicht. Man kann auch direkt aus dem Repositorium eine SQL-Query absetzen. In einem der folgenden Beispiele ist diese Vorgehensweise beschrieben. Das sollte nach Möglichkeit umgangen werden, sicherheitstechnisch birgt diese Funktionalität allerlei Gefahren, die Extbase sonst vermeiden würde.

An dieser Stelle möchte ich einen Blick auf das FoodOrder-Repositorium im Bestellservice werfen. Die anderen Repositorien sind uninteressant, weil Sie mit den Standardmethoden, die ein Extbase-Repositorium bietet, auskommen. Das FoodOrder-Repositorium hat eine Sonderstellung, weil in einer DB-Tabelle sowohl die aktuellen Bestellungen, als auch die veralteten, geänderten gespeichert werden. Die Frage, ob das der richtige Weg sei, oder man zusätzlich noch ein Archiv-Objekt für Bestellungen anlegt, habe ich mir dabei natürlich auch gestellt. Da ich bisher noch keine Erfahrungen mit Domain-driven Design hatte, konnte ich sie mir leider nicht genau beantworten. Wegen Redundanzen im Datenbankmodell und unnötigen Kopieraufwandes habe ich jedoch auf ein zusätzliches Ob-

jekt verzichtet. Allerdings macht es das, wie man in den folgenden Beispielen sieht, nicht einfacher.

```
01   public function findLatestBySupplyPointAndDeliverydate(
          Tx_DitFood_Domain_Model_SupplyPoint $supplyPoint,
          Tx_DitBase_Utility_DateTime $deliverydate){
02      $query = $this->createQuery();
03      $result = $query->matching(
04           $query->logicalAnd(
05              $query->equals('supplyPoint', $supplyPoint),
06              $query->equals('deliverydate', $deliverydate)
07           )
08        )
09        ->setOrderings(array('orderdate' =>
             Tx_Extbase_Persistence_QueryInterface::ORDER_DESCENDING))
10        ->setLimit(1)
11        ->execute();
12      $object = NULL;
13      if (count($result) > 0) {
14         $object = current($result);
15      }
16      return $object;
17   }
```

Die dargestellte Funktion soll die aktuelle Bestellung anhand der Lieferstelle und des Lieferdatums finden. Dazu wird in Zeile 02 ein Query-Objekt erstellt. Dieses Objekt besitzt einige Methoden, mit denen man Vergleiche (Zeilen 05 und 06) und logische Operatoren (Zeile 04) nachbilden kann. Auch die Sortierreihenfolge (Zeile 09) und eine Beschränkung der Objekte (Zeile 10) des Ergebnissatzes können festgelegt werden. Im Beispiel werden alle Objekte gesucht, deren Lieferstelle UND Lieferdatum den der Funktion übergebenen Objekten entspricht. Auch hier muss man sich nicht um die Konvertierung der Objekte in eine Datenbankkompatible Form kümmern. Extbase ermittelt den Wert anhand des Objekts mit dem DataMapper, der sowohl TypoScript- als auch TCA-Einstellungen berücksichtigt, automatisch. Das erleichtert dem Entwickler die Arbeit und bringt Konsistenz in die Schnittstelle zwischen Objekteigenschaften und Datenbankfeldern. Ich habe zwar mit Zeile 09 angewiesen, dass die Bestellungen nach dem Bestelldatum abwärts sortiert werden sollen und mit Zeile 10, dass nur der erste Ergebnisdatensatz zurückgegeben werden soll, allerdings liefert die execute()-Methode (Zeile 11) immer ein Array mit den gefundenen Objekten zurück. In diesem Fall entweder ein leeres Array, oder eines mit genau einem enthaltenen Objekt. Um

also die aktuelle Bestellung zurückgeben zu können, wird in den Zeilen 12-15 noch das gefundene Objekt aus dem Ergebnisarray herausgelöst und zurückgegeben. Bei dieser Methode nehme ich an, dass als aktuelle Bestellung die bezeichnet werden kann, welche zu einem bestimmten Lieferdatum von allen gefundenen die späteste Bestellzeit hat. Da bei jeder Neu- bzw. Änderungsbestellung die aktuelle Serverzeit als Bestellzeit gesetzt wird, kann ich davon ausgehen, dass solange die Serverzeit nicht geändert wird, die Annahme zum erwarteten Ergebnis führt.

Das war eine noch recht einfache Übung. Leider ist es im folgenden Beispiel nicht möglich, die $query-Methoden zu nutzen. Extbase unterstützt keine SQL-spezifischen Funktionen, wie z.B. MAX()[27]. Außerdem fehlt die Möglichkeit der Integration von Subselects (Unterabfragen)[28]. Deswegen greife ich hier auf die händische Variante mit `$query->statement($SqlQuery)->execute()` zurück:

Aufgabe der Funktion: Alle Bestellungen der Lieferstellen eines gegebenen Kunden in einer durch Start- und Enddatum gegebenen Zeitspanne sollen ermittelt werden.

```
01. public function findByCustomerAndDeliverydateRange($customer,
              $minDeliverydate, $maxDeliverydate){
02.   $query = $this->createQuery();
03.   $extbaseFrameworkConfiguration =
      Tx_Extbase_Dispatcher::getExtbaseFrameworkConfiguration();
04.
05.   $result = $query->statement('SELECT DISTINCT
      tx_ditfood_domain_model_foodorder.*
06.      FROM tx_ditfood_domain_model_foodorder
07.      LEFT JOIN tx_ditfood_domain_model_supplypoint
08.         ON
      tx_ditfood_domain_model_foodorder.supply_point =
      tx_ditfood_domain_model_supplypoint.uid
09.      WHERE ((tx_ditfood_domain_model_supplypoint.customer = "' .
      $customer->getUid() . '"
10.         AND tx_ditfood_domain_model_foodorder.deliverydate >= "' .
      $minDeliverydate->getTimestamp() . '")
11.         AND tx_ditfood_domain_model_foodorder.deliverydate < "' .
      $maxDeliverydate->getTimestamp() . '")
```

27 MAX(): Eine SQL-Funktion, die aus einer Auswahl von Werten den höchsten ermittelt und zurück gibt

28 Eine SQL-SELECT-Anweisung verschachtelt in einer anderen SELECT-Anweisung

```
12.       AND tx_ditfood_domain_model_foodorder.deleted=0
13.       AND tx_ditfood_domain_model_foodorder.t3ver_state<=0
14.       AND tx_ditfood_domain_model_foodorder.hidden=0
15.       AND tx_ditfood_domain_model_foodorder.sys_language_uid IN (0,-1)
16.       AND tx_ditfood_domain_model_foodorder.pid IN (' .
     $extbaseFrameworkConfiguration['persistence']['storagePid'] . ')
17.       AND tx_ditfood_domain_model_supplypoint.deleted=0
18.       AND tx_ditfood_domain_model_supplypoint.t3ver_state<=0
19.       AND tx_ditfood_domain_model_supplypoint.hidden=0
20.       AND tx_ditfood_domain_model_supplypoint.sys_language_uid IN (0,-1)
21.       AND tx_ditfood_domain_model_supplypoint.pid IN (' .
     $extbaseFrameworkConfiguration['persistence']['storagePid'] . ')
22.       AND tx_ditfood_domain_model_foodorder.orderdate = (
23.          SELECT MAX(orderdate)
24.             FROM tx_ditfood_domain_model_foodorder FO2
25.             WHERE tx_ditfood_domain_model_foodorder.deliverydate =
                FO2.deliverydate
26.                 AND tx_ditfood_domain_model_foodorder.supply_point =
     FO2.supply_point' . /* Unterscheidung zwischen supplyPoints */ '
27.       )
28.     ORDER BY tx_ditfood_domain_model_foodorder.orderdate DESC')
29.          ->execute();
30.  return $result;
31. }
```

Das ist der Code mit der langen Query, die nötig ist, um die Aufgabe zu lösen. Ich habe sie nicht komplett selbst geschrieben, sondern mir den größten Teil der Arbeit von Extbase abnehmen lassen. Ohne den Zusatz der Zeilen 22-27, die leider elementar sind, könnte man die Query nämlich mit den 'herkömmlichen' Extbase-Mitteln lösen:

```
01. $result = $query->matching(
02.     $query->logicalAnd(
03.        $query->equals('supplyPoint.customer', $customer),
04.        $query->greaterThanOrEqual('deliverydate', $minDeliverydate),
05.        $query->lessThan('deliverydate', $maxDeliverydate)
06.     )
07.  )
08.  ->setOrderings(array('orderdate' =>
          Tx_Extbase_Persistence_QueryInterface::ORDER_DESCENDING))
09.  ->execute();
```

Ich habe diese Query einmal ausführen lassen, mir dann die generierte SQL-Anweisung kopiert und die Zeilen 22-27 eingefügt. Mit dem Subselect in diesen Zeilen wird geprüft, ob die Bestellung, die gerade auf die durch den WHERE-Teil vorgegebenen Konditionen geprüft wird, auch die höchste Bestellzeit (MAX(orderdate)) hat, die der Lieferstelle zu diesem Lieferdatum zugewiesen ist.

Diese ausgeschriebene Query hat einige Nachteile. Sie ist unübersichtlich und nicht selbsterklärend. Es ist nicht ohne weiteres möglich zu prüfen, ob die Funktion auch hält, was ihr Name verspricht, nämlich *findByCustomerAndDeliverydateRange (findeMitKundeUndLieferzeitspanne)*. Eine direkt eingetragene Query geht an den den meisten Sicherheitsschranken von Extbase vorbei. Da ich allerdings Extbase selbst den Großteil der Query habe produzieren lassen, ist das hier nicht das Problem. Allerdings könnte die Query mit zukünftigen von Extbase generierten Querys abweichen. So habe ich die Query schon einmal mit einer älteren Extbase-Version ausgeführt. Dabei wurde eine kürzere SQL-Anweisung produziert. Extbase hatte damals nämlich noch nicht die zusätzlichen Felder der verknüpften SupplyPoint-Tabelle geprüft, sodass die Zeilen 17-21 im Beispiel mit der $query->statement()-Anweisung noch nicht existierten. Die abgefragten Felder in diesen zusätzlichen Zeilen sind TYPO3-eigene Felder, mit denen man Datensätze verstecken (hidden), löschen (deleted), versionieren (t3ver_state) und lokalisieren (sys_language) kann. 'pid' (Page-ID) ist die Seiten-ID im TYPO3 Seitenbaum. Da sie mit Extbase konfigurierbar ist, wird die kommaseparierte Liste der möglichen Seiten-IDs dynamisch aus der Extbase-Framework-Konfiguration eingesetzt.

Angesichts der beschriebenen Nachteile ist es ratsam, sobald es einen Extbase-Weg für die Lösung dieser Aufgabe gibt, diesen zu nutzen. Bis dahin wird es so gehen müssen.

4.2 Controller und Actions

Mit Controllern lässt sich der Programmablauf steuern. In Extbase werden die ActionController unterstützt. Dabei setzt sich jeder Controller aus Actions zusammen. Da die Controller meistens für einzelne Domain-Objekte zuständig sind, werden sie häufig auch nach Ihnen benannt. So gibt es im Bestellservice unter anderen CustomerController, SupplyPointController und FoodOrderController.

Die Controller des Bestellservice erben vom Extbase ActionController. Das tun sie aber nicht direkt. Um projektweit Controller-Methoden zur Verfügung stellen zu können, habe ich einen BaseController angelegt. Die Actions, die von dem Controller gesteuert werden, werden von Methoden mit vorgegebener Namenskonvention repräsentiert. Hier im Beispiel eine indexAction:

```
01.  class Tx_DitFood_Controller_SupplyPointController extends
       Tx_DitFood_Controller_BaseController {
02.     /*
03.      * Index action for this controller.
04.      *
05.      * @param Tx_DitFood_Domain_Model_SupplyPoint $supplyPoint
06.      * @return string The rendered view
07.      */
08.     public function indexAction(
          Tx_DitFood_Domain_Model_SupplyPoint $supplyPoint) {
09.        $customer = $supplyPoint->getCustomer();
10.        $deliverydate =
Tx_DitFood_Domain_Service_FoodOrderService::getDefaultDeliverydateByCustomer($c
ustomer);
11.
12.        $foodOrderWillBeNew =
Tx_DitFood_Domain_Service_FoodOrderService::doesFoodOrderExistForSupplyPointAnd
Deliverydate($supplyPoint, $deliverydate) === FALSE;
13.
14.        $this->view->assign('customer', $customer);
15.        $this->view->assign('supplyPoint', $supplyPoint);
16.        $this->view->assign('deliverydate', $deliverydate);
17.        $this->view->assign('foodOrderWillBeNew', $foodOrderWillBeNew);
18.
19.        // can be omitted
20.        return $this->view->render();
21.     }
22.  }
```

In Zeile 01 wird die Klasse definiert. Sie erbt von einem übergeordneten Controller, der wiederum von Tx_Extbase_MVC_Controller_ActionController erbt. Die Action mit dem Namen *index* wird in Zeile 08 deklariert. Der Funktionsname folgt der Konvention, nämlich *{actionName}Action*. Diese Action wird ausgeführt, wenn der Extbase-Dispatcher einen Request mit den Daten "Controller = SupplyPoint" und "Action = index" erhält.

Ein sehr wichtiger Aspekt dieser Actions sind die übergebenen Parameter. Zwei Aufgaben übernimmt Extbase dabei, die ich fürderhin ein wenig ausführen möchte.

1. Bereitstellung der Parameter – Die in der Action erwarteten Parameter werden aus dem Request herausgelöst, gecastet[29] oder in ein Objekt umgewandelt und der Action übergeben.

2. Validierung der Parameter – Die übergebenen Objekte werden auf Validität geprüft.

1. Bereitstellung der Parameter

Extbase nutzt dafür die Reflection-Klassen, die PHP ab Version 5 bereitstellt. Mit deren Hilfe kann man zur Laufzeit Informationen über Klassen und Methoden herausfinden. Nicht besonders eindrucksvoll ist die Möglichkeit, die Namen der Methoden einer Klasse auszulesen, da diese Möglichkeit schon seit PHP 4 mit get_class_methods($className) möglich war. Allerdings können auch Zusatzinformationen aus den Doc Comments (Dokumentationskommentare) ausgelesen werden. Extbase holt sich die Informationen über die aufzurufende Action und die dort erwarteten Parameter. Anhand des Annotation-Tags (Anmerkungsschlagworte) @param und der darauf folgenden Typenbezeichnung erkennt es den erwarteten Parameter. Für einen simplen Datentyp wird einfach gecastet, Extbase versteht sich aber auch auf die Erstellung von Domain Models.

```
http://localhost/index.php?tx_ditfood_main[supplyPoint]=5
```

Mithilfe dieser Beispiel-URI und den obigen Daten stellt der PropertyMapper in Extbase das SupplyPoint-Objekt mit der Datenbank-ID 5 der Action als Parameter zur Verfügung.

Kann ein Parameter der Action nicht bereitgestellt werden und er ist nicht optional, kann also nicht weggelassen werden, wirft Extbase eine Exception.

29 Casten heißt in der Programmierung, dass ein Wert seinen Datentyp wechselt. So kann man beispielsweise in PHP aus der Zeichenkette „15“ eine Ganzzahl 15 machen: `$ganzzahl = (int) „15“;`

Der PropertyMapper kann auch mit noch nicht persistierten Objekten umgehen, wenn zum Beispiel ein Objekt per Formular erstellt wird und per Request die einzelnen Eigenschaften des Objektes übergeben werden:

tx_ditfood_main[newSupplyPoint][title]=&tx_ditfood_main[newSupplyPoint][customer]=2

Dann wird eben ein vorläufiges Objekt erzeugt, das der Action übergeben wird. An dieser Stelle kommt die zweite Aufgabe von Extbase beim Aufruf einer Action ins Spiel.

2. Validierung der Parameter

Um dem Entwickler Zeit und Codezeilen einzusparen, validiert Extbase die Eigenschaften von Domain Models. Dafür werden die Doc Comments der Eigenschaften des erwarteten Models ausgelesen. Hier ein Beispiel, in dem per Doc Comment festgelegt wird, dass die Eigenschaft $title des SupplyPoints nicht leer sein darf:

```
01. class Tx_DitFood_Domain_Model_SupplyPoint extends
        Tx_Extbase_DomainObject_AbstractEntity {
02.
03.   /**
04.    *
05.    * @var string
06.    * @validate NotEmpty
07.    */
08.   protected $title;
09.
10.   /* ... */
```

Für die Validierung ausschlaggebend ist hier die @validate-Annotation. In Exbase ist eine beliebig erweiterbare Liste von Validatoren hinterlegt. Dazu gehört unter anderem auch der NotEmptyValidator. Dieser prüft die Eigenschaft und erstellt bei übergebenem NULL-Wert oder einer leeren Zeichenkette einen Fehler.

Aber auch das ist noch nicht alles. Für eine komplexere Prüfung kann es nötig sein, mehrere Eigenschaften des Objektes miteinander zu vergleichen. Für solche

Fälle bietet Extbase die Möglichkeit des ObjectValidators. Er nimmt ein komplettes Objekt entgegen, damit die einzelnen Eigenschaften gegeneinander geprüft werden können. Das ist zum Beispiel dann sinnvoll, wenn eine Zeitspanne durch zwei Zeitpunkte festgelegt wird und dann geprüft werden soll, ob der eingegeben Startzeitpunkt auch wirklich vor dem Endzeitpunkt liegt.

Die Validierung von Objekten als Parameter in Actions kann mit der `@dontvalidate`-Annotation unterbunden werden.

Mit der Typenüberprüfung und der Validierung von Objekten nimmt Extbase dem Entwickler jede Menge Code ab. Code, der ansonsten die Controller mit immer wiederkehrenden Typwandlungen und Validierungskonstrukten überlaufen lassen würde. Und außerdem werden von vornherein potentielle Sicherheitslücken durch manipulierte URIs direkt vom Framework geschlossen. Extbase *vergisst* in keiner Action, jeden einzelnen Parameter auf Herz und Nieren zu prüfen.

4.3 View - Die grafische Nutzeroberfläche

Mit View möchte ich in diesem Zusammenhang die grafische Nutzeroberfläche (GUI) des Bestellservices bezeichnen. In diesem Abschnitt widme ich mich besonders den Techniken, die für die Erarbeitung und Umsetzung einer kundenorientierten Benutzerschnittstelle geeignet sind.

4.3.1 Usability

Mit Usability (Benutzerfreundlichkeit) wird die Bedienbarkeit eines Produktes, im Falle des Bestellservices einer Webanwendung bezeichnet. Kann man Usability als einen Faktor betrachten, der wesentlich zur Akzeptanz des Produktes durch den Benutzer beiträgt. Eine Website mit guter Usability erklärt sich dem Anwender von selbst. Dabei sollen einerseits unerfahrene Nutzer möglichst einfach in die Funktionsweise des Systems hereinfinden, erfahrene Nutzer andererseits sol-

len auch ihre Arbeit machen können, ohne von Hinweistexten für Anfänger erschlagen zu werden. Die Website kann dafür als Werkzeug betrachtet werden, dass sich nicht nur der Hand des Entwicklers anpasst, sondern auch in der Hand eines jeden Anwenders leistungsstark genutzt werden kann.

Usability wird von allen denkbaren Bereichen der Nutzerschnittstelle beeinflusst. Unter anderem durch:

- Grundstruktur der Seite
- Anordnung der Elemente
- Eindeutigkeit
- Wiedererkennbarkeit
- aussagekräftige Symbole
- Naming[30]
- Schriftgrößen
- etc.

Für die Usability spielen auch die Erfahrungen der einzelnen Nutzer eine wichtige Rolle. So wie es die Menschen aus der realen Welt gewohnt sind, erwarten sie es auch im digitalen Bereich. Manche Dinge haben sich im Webbereich auch etabliert, so ist z.B. die Position für Suchfunktionalität auf Webseiten zumeist oben rechts. Für gute Usability können zwar viele Faustregeln beachtet werden, aber getestet werden kann sie nur von realen Anwendern und unter echten Umgebungsbedingungen. Dafür gibt es die sogenannten Usability-Tests, bei denen das Verhalten des Benutzers auf der Webseite, aber auch seine körperlichen Reaktio-

30 Als Naming bezeichnet man den exakten Wortlaut von Beschriftungen auf der grafischen Oberfläche

nen auf bestimmte Ereignisse genau beobachtet werden. Im Falle des Bestellservices wollen wir auf umfangreiche, kostenintensive Usability-Tests verzichten, stattdessen aber in Absprache mit dem Kunden bleiben und auf Ungereimtheiten, auf die die Nutzer treffen, zeitnah reagieren. Dass ich auf Usability-Tests verzichte soll mich aber nicht davon abhalten, mir über Usability intensiv Gedanken zu machen! [dpp] (S. 227 f.)

Im folgenden Abschnitt werde ich auf Mock-Ups eingehen, die schon im frühen Projektstadium Einblick in die spätere Benutzeroberfläche bieten wird. Vorteil: Weil der View ein eigenständiges System ist, kann er unabhängig von der wirklichen Funktionalität entwickelt werden. Die Ergebnisse können bei Bedarf dann ganz einfach für das laufende System angepasst werden.

4.3.2 Entwurf der GUI mit Mock-Up

Mock-Ups (Attrappen) bezeichnen im Bereich der Softwareentwicklung den Prototypen einer Software oder eines Teils der Software. Meistens dafür gedacht weggeworfen zu werden und dem endgültigen System zu weichen, bringen sie eingeschränkte Funktionalität mit, die nur einen kleinen, ausgewählten Teil der Gesamtsoftware widerspiegelt oder ersetzt. Mit einem HTML-Mock-Up kann man in Webprojekten zum Beispiel die Anforderungen an die grafische Oberfläche prüfen. Es handelt sich dann meist um XHTML-Dokumente, die mit CSS und JavaScript angereichert sein können. Diese Mock-Ups werden manchmal auch als interaktive Prototypen bezeichnet. Auf programmiertechnischer Ebene kann man mit einem Mock-Up ein Objekt simulieren, das es noch nicht gibt, oder das zu komplex für den aktuellen Anwendungsfall ist. [php5] (S. 63)

Für den Bestellservice habe ich in einem frühen Projektstadium Mock-Ups als Klickmodelle eingesetzt. Es handelt sich dabei um statische HTML-Seiten, die einen guten Überblick über den Aufbau der Seite geben. Da ich bisher noch nicht so viel Erfahrung mit jQuery hatte, konnte ich das Mock-Up nutzen, um jQuery-

Effekte auszuprobieren, ohne dass der PHP-Hintergrund, der dann endgültig HTML und JavaScript generiert, existieren musste. Derartige Mock-Ups können zwischen Kundengesprächen erstellt, und beim nächsten Mal vorgestellt werden. Anhand des Mock-Ups können Anforderungen an die Benutzeroberfläche herausgearbeitet werden. Damit sieht der Kunde direkt, worauf er sich einlässt und kann mitbestimmen. Je mehr er darauf Einfluss nehmen kann, desto mehr steigt die Wahrscheinlichkeit, dass er langfristig zufrieden ist. Das ist ohnehin besser, als wenn er eine Lösung ungesehen präsentiert bekommt, mit der er dann einfach "leben muss". Besonders optisch orientierten Kunden hilft ein Mock-Up wesentlich mehr als ein Diagramm mit Domain Models, die miteinander abstrakt kommunizieren und gegenseitig Methoden aufrufen.

Das Gute an so einem Mock-Up: Der Kunde kann selbst schon einmal ein bisschen herumklicken, hat einen Gesamteindruck z.B. des Formulars, nimmt die kleinen Effekte wahr, die per CSS und JavaScript eingebunden wurden. Er als Domain Experte kann zusätzlich überlegen, ob damit alle Anforderungen abgedeckt werden können. Anhand der Mock-Ups kommen manchmal noch Missverständnisse an den Tag, die, je später sie entdeckt werden, umso größeren Schaden anrichten würden. Daher ist es ratsam, schon sehr früh mit Mock-Ups anzufangen. Mock-Ups sind eine vereinfachte Darstellung dessen, was der Kunde mit dem fertigen Produkt zu sehen bekommt. Sie sind oft auch hilfreich, den Kunden zu motivieren und ihm ein Gefühl für ein gelingendes Projekt zu geben.

Für den Bestellservice habe ich den Bestellprozess in ein Mock-Up umgesetzt:

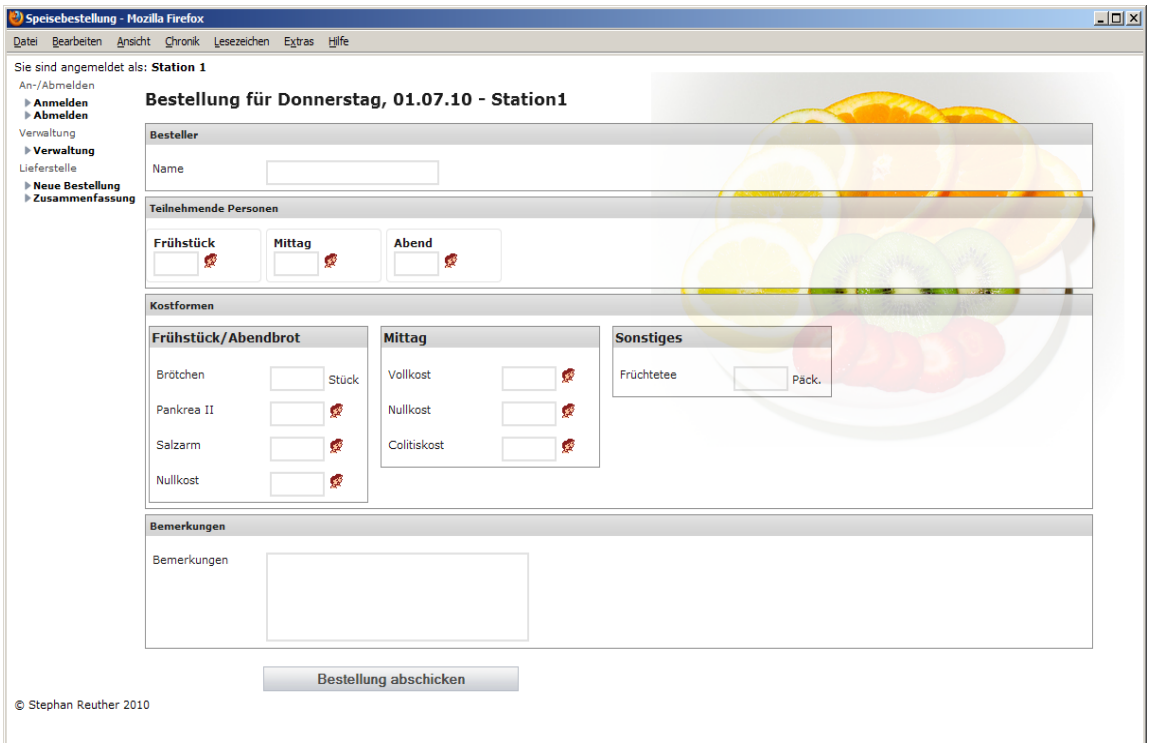


Abb. 4-2: Mock-Up - Bestellformular

Auf der ersten Seite sieht man das Bestellformular für eine Lieferstelle mit dem Namen "Station1". Das Formular ist in 4 Blöcke unterteilt, die mit verschiedenen Formularfeldern gefüllt sind. Da das Mock-Up in HTML umgesetzt ist, und nicht einfach nur ein Bild, sind die Formularfelder auch schon funktionstüchtig (Klickmodell). Dadurch kann man Beispieldaten eintragen und JavaScript-Reaktionen ausprobieren.

Hier kann man Beschränkungen des Layouts herausarbeiten:
- Was passiert, wenn man 14 verschiedene teilnehmende Personen zulässt, muss das Formular in der Anzahl also beschränkt werden. Werden dann noch die Anforderungen erfüllt (vielleicht sollen ja über 10 verschiedene teilnehmende Personen möglich sein).

- Was passiert, wenn nur eine teilnehmende Person definiert ist, wird dabei nicht sehr viel Platz verschwendet? Kommt dieser Fall bei dem Kunden vor, oder ist er ausgeschlossen.
- Wie viel Platz braucht so ein Formular? Müssen noch Optimierungen vorgenommen werden? Wenn ja, welche sind möglich?

Abb. 4-3: Mock-Up - Zielseite

Das zweite Mock-Up zeigt die Zielseite des Formulars, sie fasst die Bestellung zusammen. Zusätzlich wird hierbei gezeigt, wie sich die Anzeige bei einer Änderungsbestellung verhält. Änderungsbestellung heißt eine Bestellung dann, wenn für den Liefertag schon eine andere Bestellung vorliegt. Der Name des Bestellers, die Bestellzeit und die Werte der vorherigen Bestellung, die durch die aktuelle Bestellung geändert wurden, werden ausgegraut dargestellt. Damit können Änderungen besser nachvollzogen werden,

Je nach Anforderung kann das Mock-Up nur einen Teil, oder die gesamte Webseite umfassen.

4.3.3 Fluid

Wie in *3.3.3 Fluid* beschrieben wird Fluid in der modernen TYPO3-Extensionentwicklung als Template Engine eingesetzt. Der Vorteil von Fluid ist die Verlagerung der View-Logik direkt ins Template, sodass keine unnötigen Abhängigkeiten zwischen Controller und View entstehen, die eine reine MVC-Architektur zerstörten. In diesem Unterabschnitt möchte ich ausgewählte Lösungen vorstellen, die ich für den Bestellservice für und mit Fluid erstellt habe.

Am besten eignet sich dafür wohl das Template für das Bestellformular. Darin enthalten sind verschiedene Standard-Lösungen, die Fluid anbietet, einige einfache Erweiterungen, die man selbst implementieren kann, aber auch eine Lösung, die eine derzeitige Schwäche von Fluid auszubügeln versucht.

Das Formular muss mit zwei verschiedenen Techniken befüllt werden:

1. Die Objekteigenschaften, mit denen Fluid umgehen kann, werden objektorientiert und auf Fluid-Art gesetzt.

2. Die Objekteigenschaften, die Fluid nicht abdeckt, werden mit herkömmlichen Mitteln gesetzt.

Was das Formular erzeugen soll:

Ein valides FoodOrder-Objekt mit

- aktuell angemeldeter / vorgegebener Lieferstelle als Lieferstelle
- voreingestelltem Lieferdatum

- aktueller Bestellzeit
- Bestellername (Pflichtfeld)
- eingetragenen Notiz (Optional)
- richtige Zuordnung von Personen und Anzahl zu teilnehmende Personen
- richtige Zuordnung von Kostformen und Menge zu bestellte Kostformen

Die letzten beiden sind mit herkömmlichen Fluid-Hilfsmitteln noch nicht möglich, da es sich bei den genannten Objekten um sogenannte ObjectStorages (Objektspeicher) handelt. Fluid kann zwar Unterobjekte mit einer 1:1 Beziehung (z.B eine Kostform hat genau eine Einheit) anlegen, das Erzeugen von mehreren Unterobjekten in einer 1:n-Beziehungen (z.B. in einer Bestellung können verschiedene Kostformen bestellt werden) ist jedoch noch nicht implementiert. Deswegen musste ich einen Umweg machen, den ich hier mit beschreiben möchte.

Um das Template in den Kontext im Bestellservice einordnen zu können, will ich noch kurz auf die "umgebenden" Actions eingehen. Die newOrEditAction im FoodOrderController ruft das Template auf und übergibt die im Template benötigten Variablen. Das Formular ruft nach dem Absenden die createOrUpdateAction auf, in der im Erfolgsfall die Bestellung zusammengebaut und persistiert wird. Das Formular wird nicht nur von den Lieferstellen genutzt, sondern auch von der Küche, die auch in der Vergangenheit liegende Bestellungen ändern können.

Im Folgenden werde ich einige Zeilen aus dem Fluid-Template notieren und jeweils dazu erklären, wofür sie da sind.

```
{namespace dit=Tx_DitBase_ViewHelpers}
{namespace food=Tx_DitFood_ViewHelpers}
```

Mit den Namespace-Deklarationen legt man neue Namensräume für eigene ViewHelper an. Dieses Verfahren ist an die XML-Namespace-Deklaration angelehnt, da die Namespaces in den Fluid-Tags auch wie bei XML verwendet werden können. Die erste Deklaration zum Beispiel verweist mit dem Bezeichner "dit" von nun an auf den Ordner "/typo3conf/ext/dit_base/Classes/ViewHelpers/". Damit ist es sehr einfach möglich, ViewHelper aus beliebigen Extensions aufzurufen. Der Namensraum "f" ist standardmäßig integriert und verweist auf das Verzeichnis mit ViewHelpern, die Fluid bereitstellt.

```
<f:layout name="main" />
<f:section name="content">
   <!-- Rest des Templates -->
</f:section name="content">
```

Layouts sind selbst Fluid-Templates, die man als Vorlagen nutzen kann. In einem Layout können "Sections"[31] als Platzhalter festgelegt werden. Ich verwende das Layout mit dem Namen "main". Fluid nutzt dafür standardmäßig den Ordner "EXT:Resources/Private/Layouts/"[32] und sucht dort nach "main.html". In dem Layout ist eine Section "content" enthalten, die ich mit dem Inhalt innerhalb des SectionViewhelper-Tags fülle. Im Layout können beliebig viele Sections definert werden. Das Layout bewirkt, dass für meine Inhalte jeweils ein fester Rahmen besteht, in dem die Inhalte eingebettet sind. So muss z.B die Headerzeile nicht in jedem Template neu eingefügt, sondern kann einmal im Layout definiert werden.

```
<f:if condition="{recentFoodOrder}">
   <f:then>
      <h2>Änderungsbestellung für {supplyPoint.title} am
{deliverydate.formattedDate}</h2>
      {dit:page.title(title: 'Änderungsbestellung')}
   </f:then>
   <f:else>
      <h2>Neue Bestellung für {supplyPoint.title} am
{deliverydate.formattedDate}</h2>
   </f:else>
</f:if>
```

31 Im Layout wird eine Section mit Name als Platzhalter definiert, der dann im Fluid-Template mittels Section-Tag gefüllt werden kann

32 „EXT:“ steht hier für den Pfad zum Extensionordner

Da das Template sowohl für Neu- als auch Änderungsbestellung genutzt wird, soll auch die Überschrift angepasst werden. Über den If-ViewHelper wird geprüft, ob die von der Action übergebene Variable "recentFoodOrder" gefüllt ist. Wenn eine frühere Bestellung für den Liefertag gefunden wurde, wird die Überschrift "Änderungsbestellung" angezeigt. Aus Usability-Gründen werden noch der Titel der Lieferstelle und das Lieferdatum angezeigt. Da der Bestellvorgang auf einer Seite im TYPO3-Seitenbaum mit dem Namen "Neue Bestellung" durchgeführt wird, muss der Titel auf dieser Seite noch angepasst werden. Dies geschieht mit dem von mir erstellten Page_TitleViewHelper aus der dit_base-Extension. Dafür nutzt er die TYPO3-API. Gibt es keine bisherige Bestellung, wird als Überschrift "Neue Bestellung für ..." ausgegeben. Der Seitentitel muss nicht angepasst werden, da er direkt aus dem TYPO3-Seitenbaum genommen wird.

```
<f:render partial="formErrors" arguments="{for: 'foodOrder'}" />
```

Für den Fall, dass das Formular beim Abschicken Fehler enthält, die Extbase validiert, wird wieder dieses HTML-Template aufgerufen. Dann werden an dieser Stelle die Fehlermeldungen ausgegeben, die dem Nutzer eine detailliertere Beschreibung und Hilfe zur Fehlerbeseitigung liefern. Hier wird ein Partial eingefügt, das standardmäßig im Ordner "EXT:Resources/Private/Partials/" liegt. Durch das Attribut partial="formErrors" sucht Fluid nach einem Partial namens "formErrors.html". Ihm wird das Argument "for" mitgegeben, damit nur die aufgetretenen Fehler für das foodOrder-Objekt ausgegeben werden. Das Partial habe ich aus dem BlogExample[33] kopiert und einige Änderungen für die Lokalisierung vorgenommen. In dem Partial wird der Form_ErrorsViewHelper aufgerufen um die Fehlermeldungen darzustellen.

```
<f:form name="foodOrder" action="createOrUpdate"
   arguments="{customer: customer, supplyPoint: supplyPoint}"
   object="{foodOrder}" method="post">
      <!-- Rest des Templates -->
</f:form>
```

33 http://forge.typo3.org/projects/typo3v4-mvc/repository/show/blog_example

Das Fluid-Formtag wird beim Rendern in ein normales HTML-Formtag umgewandelt. Mit dem Attribut "action" wird die Ziel-Action bestimmt. Das ist hier die createOrUpdateAction, welche die validierte Bestellung entgegen nimmt, endgültig zusammenbaut und in der Datenbank abspeichert. Als Objekt wird dem Formular "foodOrder" mitgegeben. Dieses wird direkt von den mit Fluid möglichen Eigenschaften gefüllt (siehe: *3.3.3 Fluid - Formulare mit Fluid*)

```
<f:form.hidden property="deliverydate" value="{deliverydate
   -> dit:format.date(renderAs: 'MysqlDate')}" />
<f:form.hidden property="orderdate"
   value="{dit:format.date(renderAs: 'MysqlDatetime')}" />
<f:form.hidden property="supplyPoint" value="{supplyPoint}" />
```

Hier werden drei Hidden-Tags erstellt. Das sind Formularelemente, die im Formular nicht angezeigt werden, aber Werte beinhalten, die mit abgeschickt werden. Die ersten beiden beinhalten das Lieferdatum und das Bestelldatum. Das Bestelldatum wird hier schon mitgegeben, damit bei der Validierung des FoodOrder-Objektes kein Fehler auftritt. Es wird beim Abspeichern der Bestellung noch einmal überschrieben. Das dritte Feld definiert die Lieferstelle. Wenn eine Küche angemeldet ist, kann über dieses Feld die Bestellung eindeutig einer Lieferstelle zugeordnet werden. Für die Formatierung des Datums habe ich einen eigenen ViewHelper erstellt.

```
01. <div class="fieldset" id="besteller">
02.  <h3 class="legend">Besteller</h3>
03.  <div class="formRow">
04.     <label for="input_besteller">Name</label>
05.     <f:if condition="{dit:callStaticMethod(class:
'Tx_DitFood_Domain_Service_UserService', method: 'isKitchen')}">
06.        <f:then>
07.           <f:form.textbox property="orderer" readonly="readonly"
value="Küche" id="input_besteller" class="inputMedium" />
08.        </f:then>
09.        <f:else>
10.           <f:form.textbox property="orderer" id="input_besteller"
class="inputMedium" />
11.        </f:else>
12.     </f:if>
13.  </div>
14. </div>
```

Das ist einer der Formularblöcke (siehe *4.3.2 Entwurf der GUI mit Mock-Up*). Er wird von einem Div-Container mit der CSS-Klasse "fieldset" eingerahmt. Da ich hier nicht weiter auf die Umsetzung mit HTML und CSS eingehen möchte, werde ich der Übersichtlichkeit halber einen großen Teil des reinen HTMLs im Fluid-Template weglassen und nur die wichtigen Informationen beschreiben. Das komplette Fluid-Template befindet sich auf dem dieser Diplomarbeit beigefügten Datenträger.

Interessant ist hier Zeile 05: Mit dem IfViewHelper soll abgefragt werden, ob eine Küche angemeldet ist, oder nicht. Als condition-Attribut wird ein eigener ViewHelper angegeben.

Dafür gibt es mehrere Lösungsansätze[34]:

1. Diesen Ansatz habe ich umgesetzt. Ein generischer ViewHelper, der beim Rendern die übergebene Methode einer übergebenen Klasse statisch aufruft und das Ergebnis zurück gibt. Der Vorteil in der Nutzung dieses ViewHelpers besteht darin, dass man nicht für jede Methode eines Services einen neuen ViewHelper anlegen muss (wie bei Lösungsansatz 2). Allerdings ist der Aufruf dieses ViewHelpers auch etwas komplexer und erfordert Hintergrundwissen, nämlich wie der Service und seine Methoden heißen. Er ist damit nicht jedem Template-Autor zuzumuten.

2. Der Zweite Ansatz ist, einen isKitchenViewHelper zu erstellen. Dieser ruft den UserService auf, der zurück liefert, ob der aktuell angemeldete Nutzer zu einer Küche gehört. Die Vorteile dieser Lösung sind, dass Domainlogik nicht im Template oder im ViewHelper, sondern im Domain Service stattfindet. Ein Template-Autor nutzt diesen ViewHelper wie jeden anderen, er muss nichts von den dahin-

34 Die Lösungsansätze wurden im März 2010 von Peter Niederlag, Franz Koch und Sebastian Kurfürst diskutiert, siehe Mailing-Listen-Thread "[TYPO3-mvc] fluid: object.accessor and method with arguments"

ter liegenden programminternen Verarbeitungsweisen wissen, z.B. wie der Service heißt, der die Anfrage bearbeitet.

3. Der Controller ermittelt den Wert und übergibt dem Template das Ergebnis in Form einer Variable. Dieser Ansatz wäre möglich, würde aber noch mehr Transfer zwischen Action und Template Engine verursachen, was ich persönlich nicht gut finde, weil damit weitere Abhängigkeiten zwischen Controller und View entstehen.

Da ich selbst die Templates erstelle, habe ich die generische Variante implementiert. Und für den Ernstfall, dass ein Templateautor kommt, der das dahinter liegende System nicht kennen will, kann dann ganz schnell der isKitchenViewHelper angelegt werden.

Mit den Zeilen 05 - 12 wird folgender Effekt erzielt: Wenn die Küche angemeldet ist, soll der Bestellername schon als "Küche" vordefiniert und nicht änderbar sein. Als Lieferstelle angemeldet soll die Bestellperson ihren Namen angeben. Darum wird, je nach angemeldetem Nutzer, eine Textbox erstellt. Da sich die beiden Zweige gegenseitig ausschließen, kann ich das Attribut "id", sowie "property" zweimal mit den gleichen Werten erscheinen lassen. Gerendert wird dann ja nur ein Zweig.

Mit den verschachtelten Objekten wird das Template etwas komplizierter. Da Fluid die Erzeugung von neuen Objekten in Arrays noch nicht unterstützt, musste ich für die "teilnehmenden Personen" und die "bestellten Kostformen" einen Umweg wählen:

```
01. <div class="fieldset">
02.  <h3 class="legend"><span class="ditfood-inset">Teilnehmende
Personen</span></h3>
03.  <div class="personen">
04.  <f:for each="{customer.persons}" as="person">
05.     <div class="formGrid">
06.        <label for="person_{person.uid}">{person.title}</label>
07.        <f:form.textbox name="foodOrderArray[declaredPersons][{person.uid}]
[amount]" class="ac_input numeric" id="person_{person.uid}"
value="{dit:getMultipleObjectFormValue(stackName: 'declaredPersons',
findByProperty: 'person', identifier: person.uid, getProperty: 'amount',
```

```
multipleObjectFormValues: multipleObjectFormValues)}" />
08.        <span class="einheit">{person.unit.title ->
f:format.crop(maxCharacters:4)}</span>
09.        <f:form.hidden name="foodOrderArray[declaredPersons][{person.uid}]
[person]" value="{person.uid}" />
10.     </div>
11.   </f:for>
12.   </div>
13. </div>
```

Die multiplen Unterobjekte werden nicht direkt in das Objekt "foodOrder" gespeichert, sondern in das Array "foodOrderArray". Aber nun mal von vorn. Nach der normalen HTML-Definition wird in Zeile 04 ein ForViewHelper aufgerufen. Er iteriert durch alle möglichen Personen eines Kunden und stellt in den Kindknoten den Zugriff auf die einzelnen Personenobjekte mit {person} zur Verfügung. Nach einem Label-Tag mit der Bezeichnung der Person wird in Zeile 07 eine Textbox definiert. Weil ich nicht direkt in das Objekt "foodOrder" hinein speichere, kann ich auch nicht das Property-Attribut verwenden. Stattdessen gebe ich dem Element mit dem Attribut "name" einen Namen. Unter diesem Namen wird der eingetragene Parameter an die Zielaction geschickt. Die Zusammensetzung erfolgt so, dass das foodOrderArray sowohl für teilnehmende Personen als auch für bestellte Kostformen genutzt werden kann. Für die Wertzuweisung habe ich wieder einen eigenen ViewHelper erstellt. Er liest aus dem gegebenen, im Controller erstellten, multipleObjectFormValues-Array den Wert aus, der in das aktuelle Formularelement geschrieben werden soll. Dabei werden verschiedene Anwendungsfälle unterstützt:

1. Neubestellung: keine Werte vorausfüllen

2. Eine frühere Bestellung wird bearbeitet: Werte aus der alten Bestellung werden eingetragen

3. Bei Fehleingaben wird Formular noch einmal dargestellt: vorher eingetragene Werte werden wieder eingetragen

Dieses Verhalten erfordert ein Zusammenspiel mit dem Controller, der die Daten einzeln noch einmal aufbereiten muss. Dadurch wird der Controller unnötig aufgebläht. Die Funktionalität müsste vom Framework bereitgestellt werden. Auch die Validierung dieser Werte ist noch nicht möglich. Glücklicherweise ist das bei dem Bestellservice auch nicht nötig.

In Zeile 09 wird ein zusätzliches Hidden-Feld bereitgestellt, in dass die UID der Person gespeichert wird, um eine eindeutige Zuordnung von "Person", "Menge" zu einer "teilnehmenden Person" zu haben.

Die Kostformen werden noch einmal in ihre Kategorien unterteilt:

```
01. <div id="kategorien">
02.  <f:for each="{customer.foodformCategories}" as="foodformCategory">
03.  <div class="kategorie">
04.     <h3><span class="ditfood-inset">{foodformCategory.title}</span></h3>
05.     <f:for each="{food:getFoodformsByFoodformCategory(foodformCategory:
foodformCategory)}" as="foodform">
06.        <div class="kostformen">
07.           <div class="formRow">
08.              <label for="foodform_{foodform.uid}">{foodform.title}</label>
09.              <f:form.textbox name="foodOrderArray[orderedFoodforms]
[{foodform.uid}][amount]" class="ac_input numeric" id="foodform_{foodform.uid}"
value="{dit:getMultipleObjectFormValue(stackName: 'orderedFoodforms',
findByProperty: 'foodform', identifier: foodform.uid, getProperty: 'amount',
multipleObjectFormValues: multipleObjectFormValues)}" /> {foodform.unit.title
-> f:format.crop(maxCharacters:4)}
10.              <f:form.hidden name="foodOrderArray[orderedFoodforms]
[{foodform.uid}][foodform]" value="{foodform.uid}" />
11.           </div>
12.        </div><!-- class="kostformen" -->
13.     </f:for>
14.  </div><!-- class="kategorie" -->
15.  </f:for>
16. </div><!-- id="kategorien" -->
```

Für die Darstellung der Kostformen werden zuerst in Zeile 02 die dem Kunden zugeordneten Kategorien ausgelesen und den Kindknoten als `{foodformCategory}` zur Verfügung gestellt. Jede Kategorie hat ihren Div-Container, in den die Kostformen eingefügt werden. Die zur Kategorie zugehörigen Kostformen werden mit Hilfe des getFoodformsByFoodformCategoryViewHelpers eruiert. Der ViewHelper kapselt den Zugriff auf einen Domain-Service im Hintergrund. Der lange Name des ViewHelpers deswegen, um eine eindeutige Funktionsbezeich-

nung zu haben, die der Templateautor verstehen kann. In diesem Fall hätte ich auch den generischen ViewHelper aus dem vorhergehenden Beispiel nutzen können.

Bemerkungen

```
01. <label for="bemerkungen">Bemerkungen</label>
02. <f:if condition="{foodOrder}">
03.  <f:then>
04.     <f:form.textarea property="note" id="bemerkungen" rows="" cols="" />
05.  </f:then>
06.  <f:else>
07.     <f:if condition="{recentFoodOrder}">
08.        <f:then>
09.           <f:form.textarea property="note" value="{recentFoodOrder.note}"
id="bemerkungen" rows="" cols="" />
10.        </f:then>
11.        <f:else>
12.           <f:form.textarea property="note" id="bemerkungen" rows="" cols=""
/>
13.        </f:else>
14.        </f:if>
15.     </f:else>
16. </f:if>
```

In diesem Feld wird das Fluid-Problem bearbeitet, dass, wenn man in einem Tag das Attribut "value" füllt, bei jedem Neuladen der Seite dieser Wert eingetragen wird. Das ist meiner Meinung nach ein Bug, da im Fehlerfall nicht mehr der vom Nutzer (möglicherweise) aktualisierte, sondern wieder der Startwert Feld steht. Es könnte in Sonderfällen aber auch beabsichtigtes Verhalten sein.

Das Feld soll sich folgendermaßen verhalten:

1. Wenn eine Änderungsbestellung beauftragt wird, soll der Kommentar von der ursprünglichen Bestellung voreingetragen sein.

2. Bei einer Neubestellung soll (natürlich) noch kein Kommentar voreingetragen sein.

3. Im Fehlerfall soll der Kommentar vom Zeitpunkt des Abschickens des Formulars voreingetragen sein.

Dafür prüfe ich erst die Bedingungen ab und erstelle dann für jeden Fall das Formularfeld. Über das Objekt {foodOrder} kann man prüfen, ob das Formular schon einmal abgeschickt wurde. Es wird von der Action zur Verfügung gestellt. Es ist beim ersten Aufruf NULL (Else-Zweig) und erst ab dem zweiten ein Objekt (Then-Zweig). Im Else-Zweig wird geprüft, ob eine Neu- oder Änderungsbestellung vorliegt. Wenn ja, wird der Wert aus der vorherigen Bestellung voreingetragen, ansonsten wird ein leeres Textfeld erzeugt.

Es fällt auf, dass die Zeilen 04 und 12 identisch sind. Diese Redundanz lässt sich mit den gegebenen ViewHelpern nicht vermeiden, da immer nur auf eine Bedingung geprüft werden kann. Zur Demonstration einer Alternative habe ich eigene ConditionViewHelper angelegt, die mehrere Argumente entgegennehmen:

```
01.  <f:if condition="{dit:condition.and(condition1: recentFoodOrder,
       condition2: '{dit:condition.not(condition: foodOrder)}')}">
02.     <f:then>
03.        <f:form.textarea property="note" value="{recentFoodOrder.note}"
             id="bemerkungen" rows="" cols="" />
04.     </f:then>
05.     <f:else>
06.        <f:form.textarea property="note" id="bemerkungen" rows="" cols="" />
07.     </f:else>
08.  </f:if>
```

Zeile 01 entspricht dabei dem PHP-Konstrukt

```
if ($recentFoodOrder && !$foodOrder){
```

Die redundante Codezeile ist zwar beseitigt, allerdings ist der Code leider noch weniger übersichtlich, als bei dem Beispiel zuvor. In zukünftigen Fluid-Versionen sollen die Booleschen Vergleichsoperatoren erweitert werden, sodass dann in etwa so etwas möglich sein könnte:

```
<f:if condition="{recentFoodOrder} AND NOT{foodOrder}">
```

Ein solcher Ansatz wurde bereits in der Mailing-Liste diskutiert, aber aufgrund der Komplexität in der Umsetzung - ein Parser wäre nötig - noch nicht realisiert.

Zur Zeit können nur einfache Vergleiche zwischen maximal zwei Operanden durchgeführt werden. [mlbool]

Kommen wir wieder zurück zum Formular. Natürlich braucht es noch einen Knopf zum Abschicken:

```
<div class="buttonSet">
   <f:form.submit name="sendFoodOrder" value="Bestellung abschicken" />
</div>
```

Das ist die einfachste Übung an diesem Template. Natürlich wird auch der Absenden-Knopf per Fluid-ViewHelper eingebunden, damit der Name korrekt gerendert wird.

Das war das Formular zum Erstellen oder Ändern einer Speisebestellung durch die Lieferstelle selbst oder die zugehörige Küche. Ich gebe zu, dass dieses Fluid-Template auf den ersten Blick ein wenig verwirrend ist. Ich habe auch das komplizierteste Template gewählt, um eine Übersicht über die Probleme zu geben, die bei der Formularerstellung mit Fluid auftreten können. In den anderen Templates nutze ich dieselben oder ähnliche ViewHelper, um an Daten zu kommen und die Viewlogik dort zu implementieren, wo sie hingehört. Allerdings sind die anderen Templates wesentlich einfacher. Formulare sind eben in der Webprogrammierung ein heißes Eisen...

Ich gebe außerdem zu, dass es wesentlich elegantere Lösungen gibt, Formulare in TYPO3-Projekten umzusetzen. Besonders im Hinblick auf die Usability und die meist damit verbundene Ajax-Technologie. So gibt es eigene Extensions für Formulare und Formulardatenverwaltung, wie z.B. ameos_formidable[35] [amfo], die schon seit Jahren entwickelt werden und viele Annehmlichkeiten bieten, über die

35 ameos_formidable ist eine Formularextension des französischen Unternehmens Ameos, welche den Einsatz von Formularen in TYPO3 vereinfacht, indem diese per XML, TypoScript und HTML-Templates konfiguriert werden können. Die Extension enthält viele Features, unter anderem Ajax-Unterstützung und umfangreiche Formularvalidierung.

das in den Kinderschuhen steckende Fluid noch nicht einmal nachzudenken denkt. Wahrscheinlich wird es auch nie so weit entwickelt werden, da es sich bei Fluid dem Grunde nach um eine Template Engine, und nicht um einen Alleskönner handelt. Ich bin mir sicher, dass es gut möglich ist, die Formularlösung eines dritten Anbieters in ein Extbase-Projekt zu implementieren. Das aber könnte in einer anderen Arbeit untersucht und bewiesen werden.

Lokalisierung von Fehlermeldungen

Im Zusammenhang mit der Validierung, beziehungsweise der Ausgabe der Fehlermeldungen tritt noch ein weiteres Problem auf. Dabei geht es um die Lokalisierung, das ist die Sprachumschaltung, der Fehlermeldungen. Fehlermeldungen (Errors) werden im objektorientierten Umfeld natürlich als Objekte gehandhabt. Es gibt mehrere verschiedene Arten von Error-Objekten. Darunter ist der "normale" Error und der ValidationError (Validierungsfehler). Der normale Error hat zwei Eigenschaften:

- Message (Nachricht) und
- Code (Fehlernummer).

Der ValidationError hat zusätzlich noch die Eigenschaft

- propertyName,

in der der Name der Objekteigenschaft gespeichert wird, für die der Fehler entdeckt wurde. Leider wird die Nachricht in Extbase direkt und auf Englisch erstellt, sodass sie nicht mehr sprachlich angepasst werden kann.

Zur Lösung dieses Problems gibt es mehrere Möglichkeiten. Zum einen kann man eigene Validatoren für die Objekteigenschaften einsetzen. Im NotEmpty-Beispiel von oben erstellt man zum Beispiel einen MyNotEmpty-Validator und

baut dort die Funktionalität nach, nutzt aber Extbase' Lokalisierung anstatt einer hardgecodeten Fehlerausgabe.

Für den Bestellservice habe ich jedoch eine einfachere Lösung eingesetzt. Ich nutze das Fehlerobjekt, um aus dessen Eigenschaften mit Fluid einen eindeutigen Schlüssel zusammenzubauen, den ich dann für die Übersetzung/Lokalisierung einsetze. Das sieht im Fluid-Template dann so aus:

```
01. <f:form.errors for="{for}">
02.  <div class="error">
03.     <f:if condition="{error.propertyName}">
04.        <f:then>
05.           <p>
06.              <f:translate key="label_{for}.
{error.propertyName}">[Error]</f:translate>:
07.              <f:for each="{error.errors}" as="errorDetail">
08.                  <f:translate key="error_{for}.
{error.propertyName}_{errorDetail.code}">[Error]</f:translate>
09.              </f:for>
10.           </p>
11.        </f:then>
12.        <f:else>
13.           <p>
14.              <!-- Hierbei handelt es sich um Errors z.B. der Klasse
Tx_Extbase_Validation_Error, die keiner Property zugeordnet sind -->
15.              <f:translate
key="error_{for}_{error.code}">[Error]</f:translate>
16.           </p>
17.        </f:else>
18.     </f:if>
19.  </div>
20. </f:form.errors>
```

Dabei entstehen dann Lokalisierungsschlüssel wie:

Zeile 06: Bezeichner der Objekteigenschaft, bei welcher der Fehler aufgetreten ist

- label_foodOrder.orderer
- label_foodOrder.supplyPoint

Zeile 08: Fehlerausgabe (PropertyValidation - Extbase)

- error_foodOrder.orderer_1221560718

- error_foodOrder.orderer_1221560910

Zeile 15: Fehlerausgabe (ObjectValidation - selbst)

- error_foodOrder_1276012301

- error_foodOrder_1276064780

Den vom ObjectValidator erstellten Fehlern kann keine Objekteigenschaft zugewiesen werden. Deswegen musste ich mich bei dem Fehlerschlüssel auf das Objekt (foodOrder) und den Fehlercode beschränken. Das ist zwar nicht sehr aussagekräftig, aber damit muss ich leben. Die Fehlermeldungen werden nicht so oft angepasst, daher sollte das auch kein Problem sein.

Die Idee, die Übersetzung mittels Error-Objekteigenschaften umzusetzen, stammt von Martin Helmich [mmh] (S.82), ich habe dabei aber noch Anpassungen vorgenommen.

4.3.4 jQuery und Ajax

Auf der Grundlage von 3.3.4 jQuery – die JavaScript-Bibliothek möchte ich hier beschreiben, an welchen Stellen im Projekt ich jQuery und Ajax eingesetzt habe. JavaScript stellt für mich eine Anreicherung der Webseite um einige Effekte und Annehmlichkeiten dar. Dies gilt besonders im Hinblick auf die Usability, auf die ich in 4.3.1 Usability näher eingegangen bin. JavaScript sollte aber nicht zum tragenden Teil der Applikation werden, da die Funktionstüchtigkeit einer Website bei abgeschalteter JavaScript-Unterstützung gewährleistet bleiben sollte.

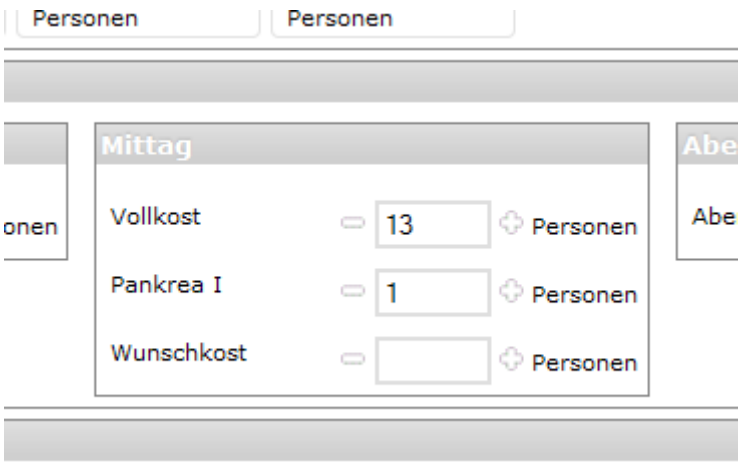


Abb. 4-4: Menge per Mausklick ändern

Ein kleines JavaScript-Feature ist das Ändern der bestellten Anzahl per Mausklick. Besonders bei Änderungsbestellungen, wo sich die Änderungen der Werte in Grenzen halten, ist es einfacher, die Werte per Klick anzupassen. Ansonsten müsste man für diese kleine Aufgabe mit der Maus zum Bearbeiten in das Feld klicken, um dann die Tastatur zur Hand zu nehmen und den Wert einzutragen. Je mehr Felder geändert werden müssen, desto nerviger würde der Wechsel zwischen Maus und Tastatur. Allerdings birgt dieses Feature auch Gefahren. Es könnte dazu avancieren, dass Werte nur noch darüber angepasst werden. Besonders bei Bestellwerten ab 20 wäre das eine aufwändige Klickerei. Zum einen habe ich die Knöpfe jeweils ausgegraut dargestellt, damit sie als Zusatzfunktion erkannt werden können. Zum anderen wird bei mehr als 10 Klicks auf ein Plus oder Minus ein Hinweistext eingeblendet, der helfen soll, das Formular möglichst effektiv zu nutzen. Bei deaktiviertem JavaScript werden keine Plus- und Minusknöpfe angezeigt. Die Funktionalität des Formulars bleibt aber erhalten.

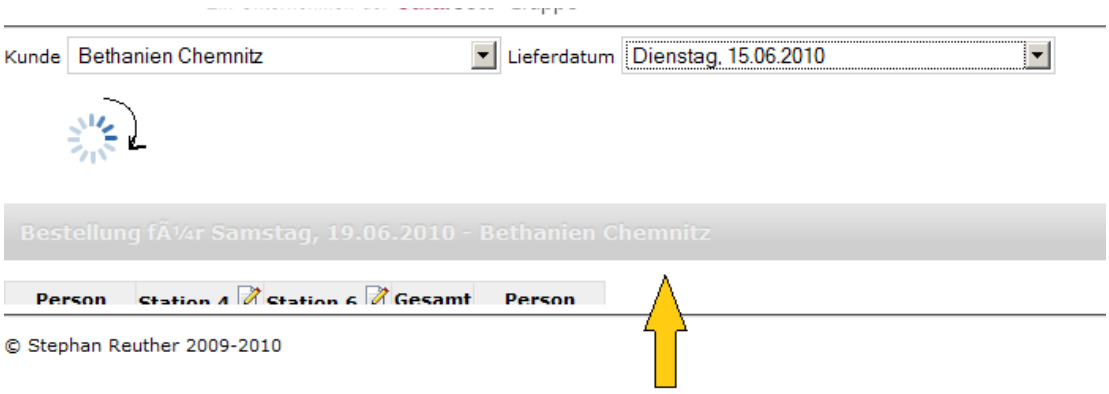


Abb. 4-5: Bestellungsübersicht per Ajax nachladen

Für die Anzeige der Bestellungsübersicht für Küche und Rechnungsstelle habe ich Ajax eingesetzt. Die Küche kann auswählen, für welchen Kunden an welchem Tag die Bestellungen der Lieferstellen angezeigt werden sollen. Die Darstellung erfolgt tabellarisch. Über der Tabelle sind zwei Select-Boxen. In der eine kann man seine zugeordneten Kunden auswählen, in der anderen stehen die Lieferdaten zur Auswahl bereit. Ändert man einen Wert, wird die Tabelle unten ausgeblendet und im Hintergrund per Ajax die neue Tabelle geladen. Zum Anzeigen der Aktivität im Hintergrund steht, für Ajax-Requests üblich, wird statt der Tabelle ein Spinner angezeigt. Wenn die Daten vom Server kommen, wird die Tabelle mit den neuen Daten eingeblendet. Der Rest der Seite wird nicht neu geladen.

Für dieses Verhalten lohnt sich ein Blick in den Quellcode, hier am Beispiel des Lieferdatums (deliverydate):

Ausgangspunkt ist hier ein Fluid-Template, das die Selectbox mit den zur Auswahl stehenden Optionen rendert.

```
<label for="deliverydatePicker">Lieferdatum</label>
<dit:form.select name="deliverydatePicker" class="deliverydatePicker"
   options="{arbitraryDeliverydates}" optionLabelField="date:l, d.m.Y"
   optionValueField="date:Y-m-d" value="{deliverydate
      -> f:format.date(format: 'Y-m-d')}" />
```

Daraus entsteht z.B. folgendes HTML:

```
<label for="deliverydatePicker">Lieferdatum</label>
<select class="deliverydatePicker" name="deliverydatePicker">
   <option value="2010-06-16" selected="selected">Mittwoch, 30.06.2010</option>
   <option value="2010-06-15">Dienstag, 29.06.2010</option>
   <option value="2010-06-14">Montag, 28.06.2010</option>
   <option value="2010-06-13">Sonntag, 27.06.2010</option>
   <option value="2010-06-12">Samstag, 26.06.2010</option>
   <option value="2010-06-11">Freitag, 25.06.2010</option>
</select>
```

Wichtig hier ist das Attribut "class", denn damit kann man das Element mittels JavaScript ansprechen. Um JavaScript-Code vom reinen HTML zu trennen, ist das mittlerweile gängige Praxis und wird mit jQuery komplett unterstützt.

Im HTML-Template ist ein Bereich deklariert, dessen Inhalt durch Ajax ausgetauscht werden soll. Also der Bereich, in dem sich die tabellarischen Daten der Speisebestellungen befinden. Aufgrund der Arbeitsweise der .load()-Funktion in jQuery müssen dabei zwei Container erstellt werden:

```
<div id="foodOrderAjaxWrapOuter">
   <div id="foodOrderAjaxWrapInner">
      <!-- Daten und Tabelle, die ausgetauscht werden -->
   </div>
</div>
```

Der zugehörige JavaScript-Code sieht so aus:

```
01. $(".deliverydatePicker").change(function(){
02.   $("#foodOrderAjaxWrapInner").slideUp('slow');
03.   $("#foodOrderAjaxWrapInner").before($("#spinner").clone().css("display",
"block").css("margin", "25px 40px"));
04.
05.   var deliverydate = $(this).val();
06.   var uri = document.location.href;
07.
08.   additionalUriParams.deliverydate = deliverydate;
09.   uri = uri + '&' + makeUriParameterString(additionalUriParams,
tx_qualifier);
10.   $('#foodOrderAjaxWrapOuter').load(uri + ' #foodOrderAjaxWrapInner',
function(){
11.      $("#foodOrderAjaxWrapInner").css('display', 'none').slideDown('slow');
12.   });
13. });
```

In Zeile 01 wird dem Element mit der CSS-Klasse "deliverydatePicker" ein Eventhandler angehangen, der abgearbeitet wird, wenn sich der Wert der Selectbox ändert. Zeile 02 bewirkt das Ausblenden des inneren Containers. Dann wird der Spinner[36] eingeblendet. Hier wird kein neuer Spinner eingebunden, sondern es wird erwartet, dass es im DOM-Baum bereits ein Element mit der ID "spinner" gibt. Dieses Element wird geklont und an der Stelle eingefügt, wo es gebraucht wird. Der Vorteil ist, dass der Spinner nicht vorgeladen werden muss und dass er im Fluid-Template zentral gestyled werden kann. In den Zeilen 05-09 wird die URI für den Ajax-Request zusammengebaut. Die Art und Weise hier ist eher roh und könnte beim Einsatz von RealURL zu Komplikationen führen, aber soll hier erst einmal ausreichen. An die aktuelle URI wird ein neuer Parameter, nämlich z.B. "&deliverydate=2010-06-16", angehangen. Wenn der Parameter mehrmals in der URI aufgetaucht ist, gilt der hinterste Parameter. Da der Parameter des aktuell angeforderten Lieferdatums hinten angehangen wird, verhält sich das System also wie erwartet. Über Zeile 10 wird der Ajax-Request abgesetzt. In den äußeren Container wird mittels .load() das Ergebnis des Ajax-Requests herein geladen. Durch die zusätzliche Angabe von "#foodOrderAjaxWrapInner" wird nur der innere Container und dessen Inhalt eingesetzt. Um umgekehrt den Effekt des Einblendens zu erhalten wird der innere Container erst versteckt und dann per jQuery.slideDown() eingeblendet.

Bei abgeschaltetem JavaScript wird neben den beiden Select-Boxen ein Button angezeigt, über den die Abfrage abgesendet werden kann. Dann wird natürlich die komplette Seite neu geladen und es gibt keinen Einblend-Effekt.

Mit diesem kleinen Exkurs in die clientseitige Programmierung mit JavaScript endet dieses Kapitel.

36 Spinner (engl. Kreisel): meist animierte Grafik, die im Zusammenhang mit Ajax auftaucht und als Indikator für Hintergrundaktivität eingesetzt wird

5 Fazit

5.1 Auswertung

Im Rahmen der Diplomarbeit wurde ein Bestellservice in Form einer TYPO3-Extension erstellt. Die schriftliche Arbeit erfasste die Entstehung des Bestellservices als Projekt in mehreren Schritten. In der Planungs- und Konzeptionsphase wurden Usecases und Wireframes erstellt, an denen die Anforderungen an den Bestellservice geprüft werden konnten.

Danach wurden die Ergebnisse der Recherchen präsentiert, die einen großen Teil der Arbeit im Vorfeld ausgemacht haben. Denn eine Extension auf Extbase-Basis kann erst geschrieben und dokumentiert werden, wenn das Framework hinreichend erfasst wurde. Intensive Auseinandersetzungen mit dem Quellcode von Extbase und Fluid waren ebenso nötig wie das Anpassen des eigenen Codes an aktuelle Änderungen im Framework und der Template Engine. Auch die Basisprinzipien und die Philosophie des Domain-driven Design wurden genannt.

Bei der Prototyperstellung konnte die Modellierung erfolgreich durchgeführt werden. Schwachstellen von Extbase im Bezug auf fehlende SQL-Unterstützung wurden erkannt. Dafür musste eine Zwischenlösung erstellt werden, die, sobald Extbase auch SQL-spezifische Funktionen unterstützt, mit diesen ausgetauscht werden muss. Die Verwendung von ActionControllern und die damit im Zusammenhang stehende Bereitstellung und Validierung von Extbase wurde am Beispiel erklärt und schließt einen Sicherheitslücke, um die sich der Extensionentwickler früher jedesmal selbst kümmern musste.

Auch die grafische Oberfläche wurde beleuchtet. Anforderungen wurden erhoben und mit den Mock-Ups wurde ein Hilfsmittel vorgestellt, das während des gesamten Projektes ein flexibler Helfer ist, um auf Änderungswünsche des Kunden exemplarisch eingehen zu können.

Probleme im Zusammenhang mit Fluid durch fehlende Unterstützung von multiplen Kindobjekten wurde erfolgreich kompensiert und ein umfangreiches Template hat das Potential, das in der Template Engine Fluid steckt, beweisen können.

Mit einem Beispiel für den Einsatz von JavaScript und Ajax ist der schriftliche Teil der Diplomarbeit ausgeklungen.

Der Bestellservice ist als Extension nutzbar und kann auf TYPO3 Systemen mit Version 4.4 eingesetzt werden.

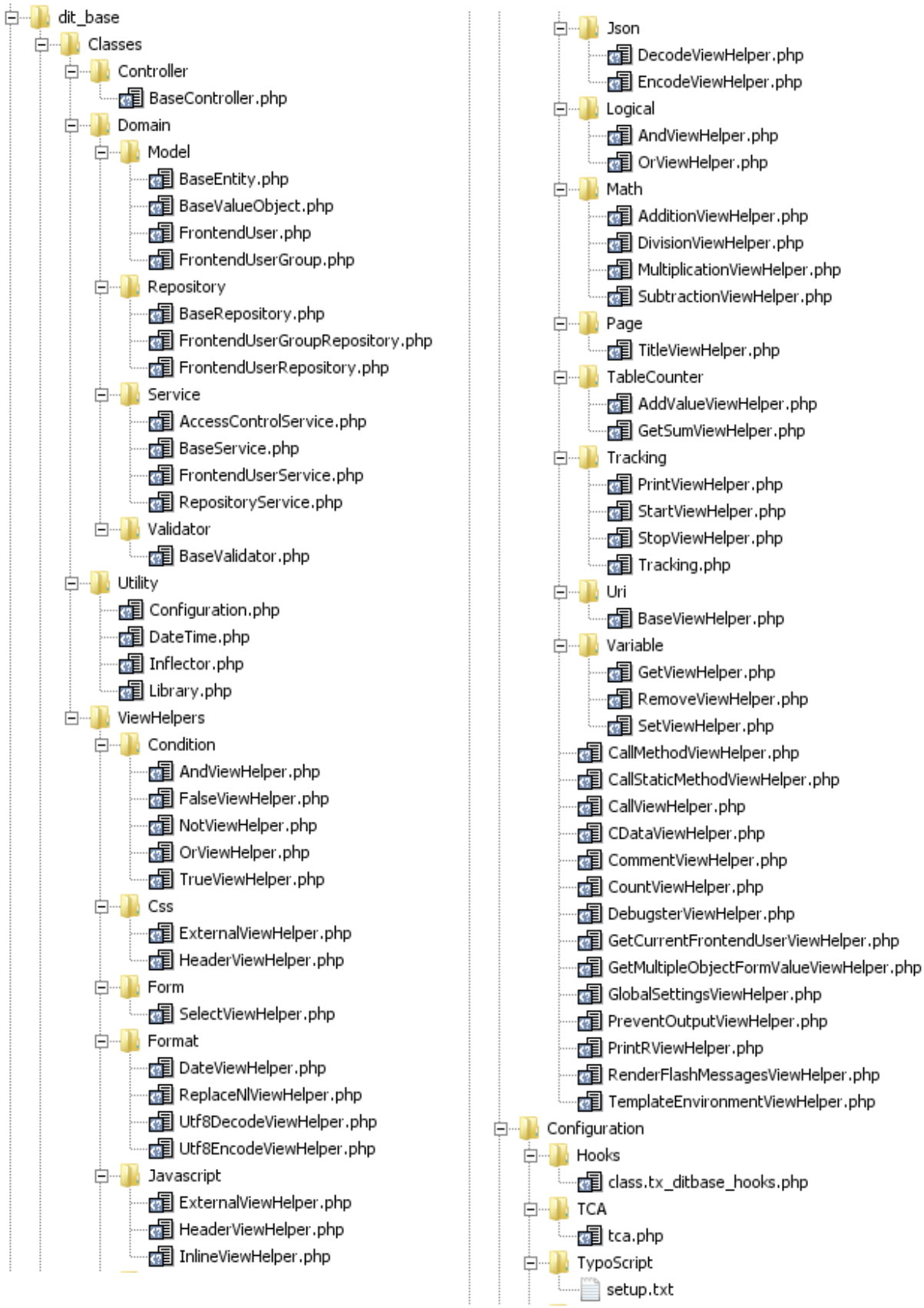
Anhang

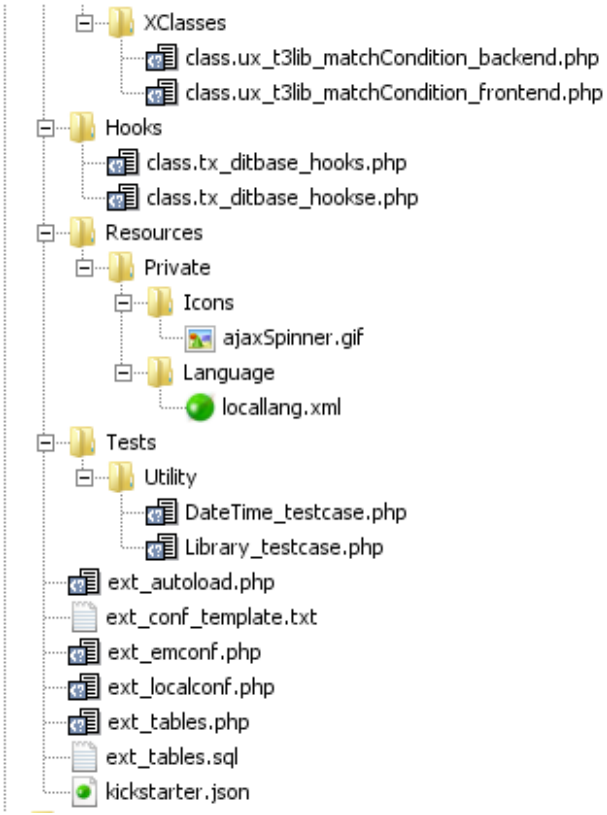
A Usecase-Tabelle

Name	Ziel	Vorbedingung	Nachbedingung	Nachbedingung im Sonderfall	Akteure	Normalablauf	Sonderfälle
#001: Login	Ein Nutzer ist eingeloggt und kann über seinen Login identifiziert werden	Nutzer ist nicht eingeloggt	Nutzer ist eingeloggt	Nutzer kann nicht eingeloggt werden und erhält eine entsprechende Nachricht	Gast / eingeloggter Nutzer	Gast klickt auf "Login" und gibt in dem Loginformular seine Logindaten ein (z.B. "Station4" / "Station4"). Nach Absenden des Formulars wird er am System angemeldet und hat nun den Status eines eingeloggten Nutzers	Logindaten werden falsch eingegeben. Gast bleibt auf der Loginseite und erhält eine entsprechende Nachricht.
#002: Logout	Nutzer ist nicht mehr eingeloggt und hat den Status "Gast"	Nutzer ist eingeloggt	Nutzer ist nicht eingeloggt		eingeloggter Nutzer / Gast	Nutzer klickt auf Logout. Er wird daraufhin vom System ausgeloggt und hat nun den Status eines Gastes	
#003: Lieferstelle führt eine neue Bestellung durch	Eine neue Bestellung ist im System, auf welche Küche und Rechnungsstelle zugreifen können	(1) eingeloggter Nutzer ist eine Lieferstelle (2) es ist gerade Bestellzeit (3) es liegt noch keine Bestellung der Lieferstelle für das Lieferdatum vor	Für das Lieferdatum existiert eine Bestellung, die der Lieferstelle des eingeloggten Nutzers zugeordnet ist	Es existiert keine Bestellung für Lieferdatum && Lieferstelle	Lieferstelle	- Lieferstelle navigiert zu "neue Bestellung" - Lieferstelle füllt ordnungsgemäß Formular aus, d.h. mindestens Pflichtfelder ausfüllen - Lieferstelle klickt auf "Bestellung verbindlich absenden" - Es erscheint eine Zusammenfassung der Daten der Bestellung - Die Daten werden im System eingetragen, sodass Küche und Rechnungsstelle darauf zugreifen können	(1) Lieferstelle gibt Daten nicht korrekt ein (d.h Pflichtfelder nicht ausgefüllt, falsche Datentypen). Folge: Formularansicht mit Fehlermeldungen und Möglichkeit zur Korrektur der Falscheingaben (2) Lieferstelle bricht Bestellprozess vorzeitig ab, indem im Menü auf andere Seite navigiert wird, bevor Bestellprozess abgeschlossen ist
#004: Lieferstelle will a) aktuelle Bestellung oder b) eine Bestellung der letzten Tage ansehen	Lieferstelle sieht übersichtliche Ansicht a) der aktuellen Bestellung oder b) einer Bestellung der vergangenen 7 Tage	(1) eingeloggter Nutzer ist eine Lieferstelle (2) es liegt eine aktuelle Bestellung vor			Lieferstelle	a) - Lieferstelle bekommt auf einer Übersichtsseite die jeweils aktuelle Bestellung angezeigt (abhängig vom Lieferdatum) b) - Lieferstelle navigiert zu "Ausgangsbuch" - dort können Bestellungen der vergangenen 7 Tage ausgewählt und übersichtlich angezeigt werden	
#005: Lieferstelle will aktuelle Bestellung bearbeiten	Lieferstelle hat aktuelle Bestellung verändert. Diese ist der Küche und der Rechnungsstelle in veränderter Form zugänglich	(1) eingeloggter Nutzer ist eine Lieferstelle (2) zu bearbeitende aktuelle Bestellung existiert (3) es ist gerade Be-	Für das Lieferdatum existiert eine Bestellung, die der Lieferstelle des eingeloggten Nutzers zugeordnet ist und die den Ände-	Ursprüngliche Bestellung der Lieferstelle bleibt erhalten, Änderungen wurden nicht vorgenommen	Lieferstelle	- Lieferstelle navigiert zu "Bestellung ändern" - wählt die zu ändernde Bestellung aus - sieht Formular mit ursprünglicher Bestellung, aber keine Bestellperson eingetragen - passt Formular entsprechend an und schickt es ab - sieht dann wie bei Neubestellung die Zusammenfassung der Bestellung. Dabei sind die geänderten Werte sichtbar hervorgehoben.	siehe Sonderfälle #003

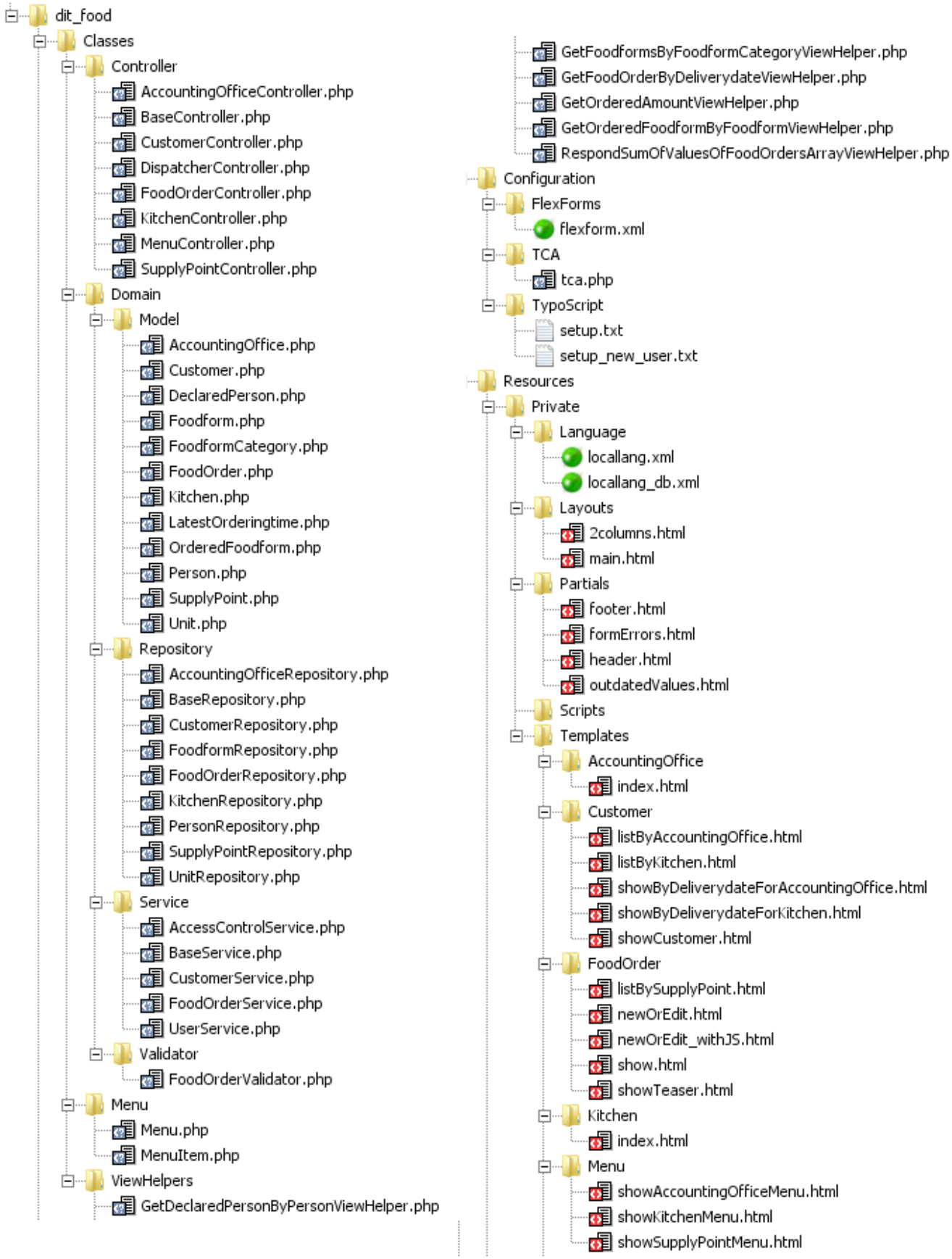
Name	Ziel	Vorbedingung	Nachbedingung	Nachbedingung im Sonderfall	Akteure	Normalablauf	Sonderfälle
		stellzeit	rungen der Lieferstelle entspricht				
#006: Küche will aktuelle Bestellungen der Lieferstellen eines Kunden sehen	Küche sieht übersichtliche Ansicht der aktuellen Bestellungen der Lieferstellen eines Kunden	(1) eingeloggter Nutzer ist eine Küche			Küche	- Küche navigiert zu "Übersicht" - wählt gegebenenfalls den Kunden in einer Auswahlbox aus - in Auswahlbox sind Daten der vergangenen Bestellungen auswählbar, aktuelles Lieferdatum ist vorausgewählt - durch Änderung des Kunden oder des Datums ist die Liste beliebig veränderbar	
#007: Küche will aktuelle Bestellungen der Lieferstellen eines Kunden ausdrucken	Küche hat einem Drucker den Druckbefehl für die Speisebestellung eines bestimmten Lieferdatums eines bestimmten Kunden gegeben.	(1) #006	Drucker der Küche hat Befehl für Bestellungsdruck erhalten		Küche	- Küche wählt mittels #006 die zu druckende Bestellung aus - klickt auf: "Bestellung ausdrucken" - Fenster öffnet sich mit Druckansicht und einem Confirm-Button, der, wenn gedrückt, den Usecase beendet und wenn Drucker korrekt konfiguriert ist, auch den Druckvorgang auslöst	
#008: Küche will beliebige Bestellung einer Lieferstelle eines Kunden bearbeiten	Küche hat beliebige Bestellung verändert. Diese ist der Küche und der Rechnungsstelle in veränderter Form zugänglich	(1) #006	Für das Lieferdatum existiert eine Bestellung, die den Änderungen der Küche entspricht	Ursprüngliche Bestellung der Lieferstelle bleibt erhalten, Änderungen wurden nicht vorgenommen	Küche	- Küche wählt mittels #006 die Bestellung aus, welche die Lieferstelle mit der zu ändernden Einzelbestellung enthält - Küche klickt auf "bearbeiten" - sieht Formular mit ursprünglicher Bestellung, welches leicht angepasst ist (Bestellperson ist voreingestellt: 'Küche' und ist nicht änderbar) - passt Formular entsprechend an und schickt es ab - Übersicht wie bei Neubestellung und Bestätigung nötig	siehe Sonderfälle #003
#009: Rechnungsstelle will monatliche Bestellübersicht sehen	Rechnungsstelle sieht Bestellungsübersicht eines Kunden pro Monat	(1) eingeloggter Nutzer ist eine Rechnungsstelle			Rechnungsstelle	- Rechnungsstelle navigiert zu "Bestellungsübersicht" - wählt einen Kunden und einen Monat aus - sieht Liste mit den Daten der Bestellungen und zusätzlichen Spalten und Zeilen mit Summen und Produkten (Definition nötig)	
#010: Rechnungsstelle will monatliche Bestellübersicht drucken	Rechnungsstelle druckt Bestellungsübersicht eines Kunden pro Monat	(1) #009	Drucker der Rechnungsstelle hat Befehl für Bestellungsdruck erhalten		Rechnungsstelle	- Rechnungsstelle verfährt nach #009 - klickt auf "Bestellübersicht drucken" - Fenster öffnet sich mit Druckansicht und einem Confirm-Button, der, wenn gedrückt, den Usecase beendet und wenn Drucker korrekt konfiguriert ist, auch den Druckvorgang auslöst	

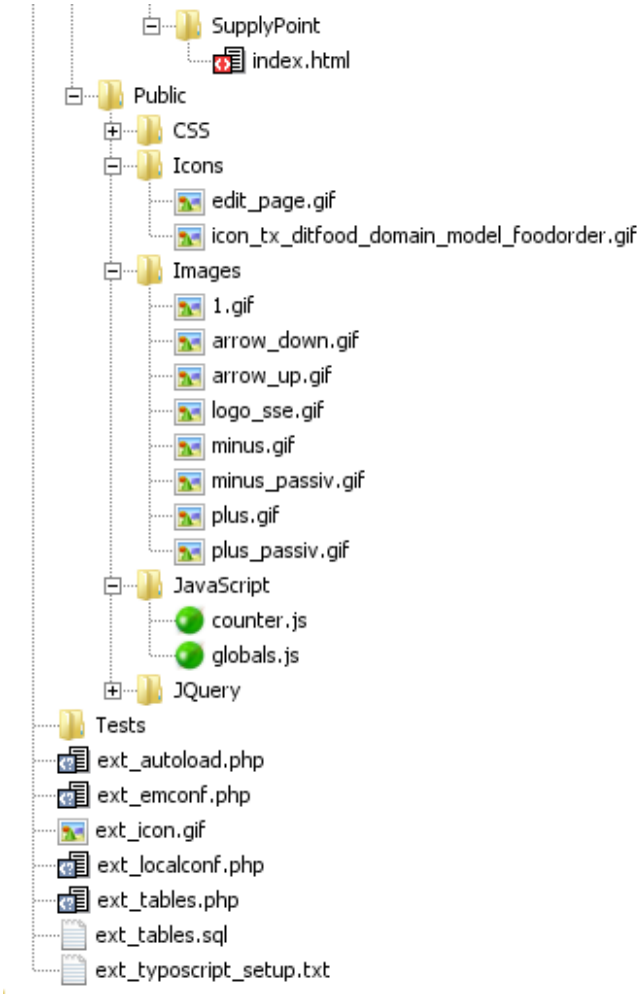
B Ordner- und Dateistruktur: dit_base




XClasses
class.ux_t3lib_matchCondition_backend.php
class.ux_t3lib_matchCondition_frontend.php
Hooks
class.tx_ditbase_hooks.php
class.tx_ditbase_hookse.php
Resources
Private
Icons
ajaxSpinner.gif
Language
locallang.xml
Tests
Utility
DateTime_testcase.php
Library_testcase.php
ext_autoload.php
ext_conf_template.txt
ext_emconf.php
ext_localconf.php
ext_tables.php
ext_tables.sql
kickstarter.json

C Ordner- und Dateistruktur: dit_food




SupplyPoint
index.html
Public
CSS
Icons
edit_page.gif
icon_tx_ditfood_domain_model_foodorder.gif
Images
1.gif
arrow_down.gif
arrow_up.gif
logo_sse.gif
minus.gif
minus_passiv.gif
plus.gif
plus_passiv.gif
JavaScript
counter.js
globals.js
JQuery
Tests
ext_autoload.php
ext_emconf.php
ext_icon.gif
ext_localconf.php
ext_tables.php
ext_tables.sql
ext_typoscript_setup.txt

Literaturverzeichnis

[so07] **Erstellen eines Use Case**. URL: http://www.iste.uni-stuttgart.de/se/teaching/courses/est1/Handouts/Spezifikation.pdf (Stand 25.06.2010)

[dpp] David Hunt: **Der pragmatische Programmierer**. - München: Hanser, 2003

[php5] Matthias Kannengießer: **Objektorientierte Programmierung mit PHP5**. - Poing: Franzis Verlag GmbH, 2007

[jak] **Handbuch der Webnavigation**. URL: http://books.google.de/books?id=b2aUsZOduacC&pg=PA259&dq=wireframes&hl=de&ei=ISwbTPlhhZf9BrCr0I4J&sa=X&oi=book_result&ct=result&resnum=2&ved=0CDYQ6AEwAQ#v=onepage&q=wireframes&f=false (Stand 25.06.2010)

[eeddd] Eric Evans: **Domain-driven Design**. - Boston: Addison-Wesley, 2004

[t3mddd] **Domain Driven Design - a brief introduction**. URL: http://www.typo3-media.com/blog/domain-driven-design-introduction.html (Stand 25.06.2010)

[woul] **Ein gewöhnliches Webprojekt**. URL: http://www.jacobsen.no/anders/blog/archives/images/project.html (Stand 01.06.2010)

[slide] **Get into the FLOW with Extbase**. URL: http://www.slideshare.net/skurfuerst/get-into-the-flow-with-extbase-and-typo3-43 (Stand 01.06.2010)

[aoeddd] **Domain-driven Design heißt eine domänenspezifische Sprache und ein Domänenmodell**. URL: http://www.aoemedia.de/online-applications/techniken/domain-driven-design.html (Stand 01.06.2010)

[t3n] Verschiedene Autoren, T3N - Open Source & Web, 2009

[t3sr] **TYPO3Wiki**. URL: http://wiki.typo3.org/index.php/De:What_is_needed_to_run_TYPO3 (Stand 01.06.2010)

[phpoo] **PHP 5 aus erster Hand**. URL: http://books.google.de/books?id=AHFNU8EUqmQC&lpg=PA99&ots=2u4a9C8_XM&dq=objektorientierung%20in%20php%20seit%20version&pg=PA99#v=onepage&q=objektorientierung%20in%20php%20seit%20version&f=false (Stand 01.06.2010)

[f3org] **FLOW3 - TYPO3's PHP Application Framework**. URL: http://flow3.typo3.org/ (Stand 25.06.2010)

[extpod] **Podcast über Extbase**. URL: http://castor.t3o.punkt.de/files/mvc2_turboH264.mp4 (Stand 01.06.2010)

[slidefluid] **Fluid - The Zen of Templating**. URL: http://www.slideshare.net/skurfuerst/fluid-the-zen-of-templating (Stand 25.06.2010)

[fluidpod] **Podcast über Fluid**. URL: http://castor.t3o.punkt.de/files/fluid.m4v (Stand 01.06.2010)

[dom] **COM-Komponenten-Handbuch**. URL: http://books.google.de/books?id=37e2VmWkGpIC&pg=PA280&dq=document+object+model&hl=de&ei=Kt0gTNT-cNJmlsQbXhYDlDg&sa=X&oi=book_result&ct=result&resnum=3&ved=0CDUQ6AEwAg#v=onepage&q=document%20object%20model&f=false (Stand 25.06.2010)

[extblog] **How to Effectively use the Repository and Query Object of Extbase?**. URL: http://blog.typoplanet.de/2010/01/27/the-repository-and-query-object-of-extbase/ (Stand 25.06.2010)

[extblogpers] **A walk through the persistence layer of Extbase**. URL: http://blog.typoplanet.de/2009/07/16/a-walk-trough-the-persistence-layer-of-extbase/ (Stand 25.06.2010)

[mlbool] **Mailingliste: Multiple conditions with logical and/or?**. URL: http://lists.typo3.org/pipermail/typo3-project-typo3v4mvc/2010-January/002908.html (Stand 25.06.2010)

[amfo] **TYPO3 Formidable - Rapid Application Development Framework for TYPO3**. URL: http://formidable.typo3.ug/ (Stand 25.06.2010)

[mmh] **Entwicklerdokumentation - Extensionentwicklung in TYPO3 4.3**. URL: http://www.mittwald.de/fileadmin/pdf/extbase_fluid.pdf (Stand 01.06.2010)

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Bearbeitungsort, Datum	Unterschrift